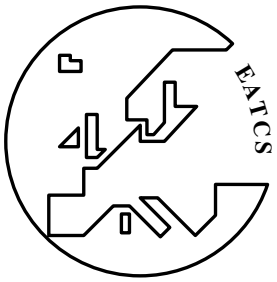
ISSN 0252–9742

Bulletin

of the

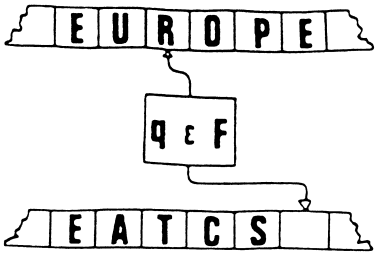
European Association for Theoretical Computer Science

EATCS



Number 141 October 2023

Council of the European Association for Theoretical Computer Science



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

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

Past Presidents:

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

Secretary Office:	Dmitry Chistikov	UK
	Efi Chita	Greece

EATCS Council Members

email addresses

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

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

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

All contributions are to be sent electronically to

bulletin@eatcs.org

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

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

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

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

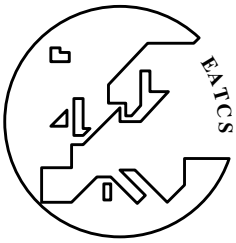
Deadlines for submissions of reports are January, May and September 15th, respectively for the February, June and October issues. Editorial decisions about submitted technical contributions will normally be made in 6/8 weeks. Accepted papers will appear in print as soon as possible thereafter.

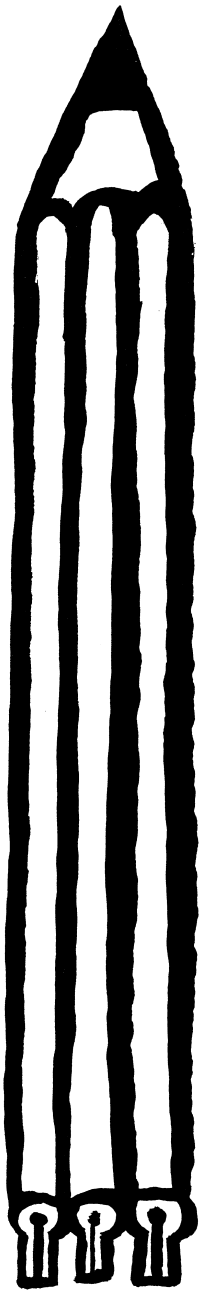
The Editor welcomes proposals for surveys, tutorials, and thematic issues of the Bulletin dedicated to currently hot topics, as well as suggestions for new regular sections.

Table of Contents

News and Conference Reports

EATCS Matters





Dear EATCS members,

I hope that you had a good summer break and that you are getting ready for the challenges awaiting you in the new academic year. This has been a great year with many activities in theoretical computer science. At the same time, I am troubled and deeply saddened by the enormous loss of life in Israel and Gaza, in Ukraine, and all the wars around us; let me express my deep wishes for peace.

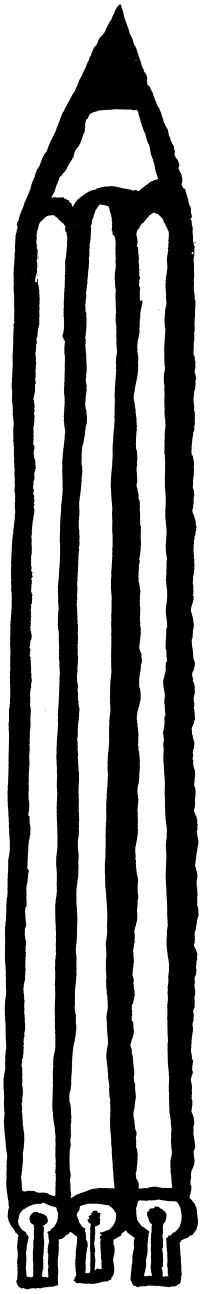
This has been an exciting summer for theoretical computer science. As one of the highlights, ICALP 2023, the EATCS flagship conference, took place this July in Paderborn. As always, the conference had an impressive scientific program highlighting the strength of the research across many areas within theoretical computer science. On behalf of the entire community and the EATCS I would like to thank the Programme Committee led by the chairs Uriel Feige and Kousha Etessami, the organizers Paderborn, led by Sevag Gharibian, and especially - all the participants, for their fantastic efforts that helped to make the ICALP 2023 conference a great success. A detailed report of ICALP 2023 is available on the pages of this issue of the Bulletin.

We also had very successful three EATCS partner conferences this summer: ESA 2023 in Amsterdam, MFCS 2023 in Bordeaux, and DISC 2023 in L'Aquila, Italy.

In the next months, you will see the calls for nominations for the EATCS Award, the Presburger Award, the EATCS Distinguished

Dissertation Award, the EATCS Fellows, and some joint awards: the Gödel Prize, the Alonzo Church Award, and the Dijkstra Prize. As usual, we are lucky to have very strong committees for each of the awards, and I thank all the award committee members in advance for their important service. I strongly encourage you to send nominations for these prestigious awards. I am aware of the fact that we are all very busy and that it takes time and efforts to prepare strong nominations, but our best researchers and best papers can only win awards if they are nominated. Moreover, awards put areas of, as well as inspirational figures in, theoretical computer science in the spotlight and can serve to inspire young researchers. I look forward to seeing who the award winners will be and to working with all of you to make the EATCS even more influential than it already is.

As usual, the October issue of the Bulletin has the first Call for Papers for ICALP 2024, the flagship conference of the EATCS and an important meeting of the theoretical computer science community world-wide. The 51st EATCS International Colloquium on Automata, Languages, and Programming (ICALP 2024) will be held on July 8-12, 2024 (workshops on July 7) in Tallinn, Estonia (https://compose.ioc.ee/icalp2024/). The conference chair is Pawel Sobocinski. We have a great list of invited speaker and expect a fantastic scientific program selected by the PCs led by the chairs Karl Bringmann and Ola Svensson (track A) and Martin Grohe (track B). ICALP 2024 will be collocated with LICS (39th Annual ACM/IEEE Symposium on Logic in Computer Science) and



FSCD (9th International Conference on Formal Structures for Computation and Deduction). Please pencil these dates in your diary and I hope to see many of you attending the next ICALP in Tallinn.

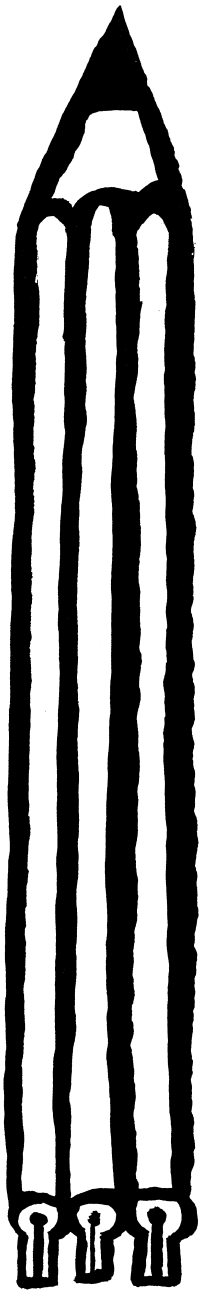
The EATCS Council decided that ICALP 2025 will be held in Aarhus, Denmark, July 8-11, 2025. The conference chairs are Ioannis Caragiannis and Kasper Green Larsen.

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, in-person or 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

October 2023



Dear EATCS member!

In front of you is the 141st Bulletin!

In this October edition, we interview Michał Pilipczuk who shares with us many experiences from his career as well as suggestions and thoughts, for example about the main challenges and opportunities for theoretical computer in the near future. Ulrike Schmidt-Kraepelin, in her Viewpoint Column, discusses the impostor phenomenon, a wide-spread problem in academia, especially in underrepresented groups and early-career researchers, and appeals to the community to help mitigate this phenomenon, making concrete suggestions.

In the Theory Blog Column, Bill Gasarch answers our questions on his experience writing for a theory blog with a very large and engaged community of readers, also highlighting two posts, one related to open problems in mathematics and another on using SAT solvers to get concrete bounds on extremal combinatorics problems.

The Distributed Computing Column features Dean Leitersdorf who won the 2023 Principles of Distributed Computing Doctoral Dissertation Award and whose work on sparse matrix multiplication has led to several breakthroughs.

In the Computational Complexity Column, Ian Mertz invites us to dip our toes into the sometimes paradox world of reusing space, discussing some highlight results in the area and presenting a wide variety of open problems. In the Logics Column, Yuri Gurevich and Andreas Blass explain the ket notation introduced in quantum mechanics.

The Formal Language Theory Column presents a historical review of the International Conference of Descriptional Complexity of Formal Systems (DCFS) which encompasses all aspects of descriptional complexity, both in theory and application, and discusses outstanding topics.

The Bulletin further includes, among other, the laudation for Boaz Patt-Shamir, the winner of the SIROCCO Prize for Innovation in Distributed Computing, an obituary for Gregory Tseytin, as well as the report on ICALP 2023.

Stefan Schmid, Berlin
October 2023

Obituary
Gregory Tseytin

On August 27, 2022, Gregory Tseytin, a brilliant Russian and American logician and computer scientist, passed away[1]. Tseytin was born in Leningrad, USSR (now St. Petersburg, Russia) on November 15, 1936. His mother was an engineer economist and his father a teacher of mathematics. Both parents came from large Jewish families in Belarus. During the war, the mother and son were evacuated to the Kirov (now Vyatka) region. Gregory was a child prodigy. In 1945, when Gregory entered the fifth grade, a committee, comprising renowned mathematicians G. M. Fichtenholz, V. A. Tartakovsky, and D. K. Faddeev, concluded that "Gregory's development in mathematics is extraordinary. He is fluent in the material taught in high school ..." At age 9, Gregory won the first prizes in the high school competitions in mathematics and physics. At school, he showed great interest in languages. He was also excellent at skating. In 1951, he graduated from high school with the highest honors but, because of his age, he was not admitted to Leningrad State University (LSU). So he just audited the lectures. In September 1952, with the permission of the Ministry of Education and supported by the LSU rector A.D. Alexandrov, he was admitted as a second-year student.

Those around Gregory were pleasantly surprised by his youthful innocence and his thirst for contacts. But it was difficult for him to socialize with his classmates. He was younger than his fellow students but knew much more. His mother accompanied him to the university and waited for him there until the classes were over. Also, he adhered strictly to formal rules in speech and actions. Eventually he adjusted to student life and became very active in the student scientific society, published a student mathematical journal, took part in organizing an English club, released the first rotaprint collection of student and tourist songs, and worked at student construction sites. In the meantime, he also graduated from an adult music school. Later, in 1960, he was among the organizers and teachers of the Youth Mathematical School.

[1]A version of this obituary was published in the Russian journal Успехи Математических Наук 78:3 (2023) 170–176. (The journal used to be translated until recently as *Russian Mathematical Surveys*).

In his second university year, Gregory attended Andrei A. Markov Jr.'s lectures on the theory of algorithms and realized that he found his teacher. Mathematical logic suited his way of thinking like no other area. His first mathematical work "On the number of steps in an algorithm" was completed by the end of the second year. Probably, this student result became part of the paper [5]. Tseytin's diploma thesis "Associative calculi with undecidable equivalence problems" was published in Reports of the Academy of Sciences [1, 2].

A few words should be said about the history of this work. In 1947, Markov and an American mathematician Emil Post independently established the algorithmic unsolvability of the word problem for finitely presented semigroups. The problem was formulated by A. Thue in 1914, long before the concept of algorithm was formalized. Markov's unsolvable Thue system was given by 13 generators and 33 relations. Tseytin's system had 5 generators and 7 relations:

$$ac \iff ca \qquad ad \iff da$$
$$bc \iff cb \qquad bd \iff db$$
$$ce \iff eca \qquad de \iff edb$$

$$cca \iff ccae$$

To this end, Tseytin used a group with an unsolvable word problem, the existence of which had been established shortly before by a Russian mathematician Pyotr S. Novikov. For more than a decade, Tseytin's example remained the simplest example of an undecidable Thue system. Eventually, Tseytin's ideas were used to construct other simple examples of relevance to computer science.

In 1956, Tseytin received his master degree in mathematics and entered the postgraduate course under Prof. Markov. For some time he worked in constructive mathematics of the Markov–Shanin school. One of Tseytin's significant achievements was proving the non-existence of a constructive real function on an interval that takes values of different signs at its ends but has no constructive root on this interval. On the other hand, he also proved that there is no algorithm that, for any constructive function with the same property, finds a constructive root in the interval.

Gregory Tseytin with his academic advisor Andrei Markov Jr. (1960)

Another remarkable Tseytin discovery was the first nontrivial positive result in this area that consisted in the fact that any constructive mapping of a constructive complete

separable metric space into another constructive metric space is continuous (in constructive mathematics, continuity implies the existence of an algorithm that finds δ from ε). This remarkable theorem was a fundamental strengthening of an earlier result by Prof. Markov on the absence of constructive discontinuities of constructive functions.

In 1959, Tseytin started as a research associate at the Institute for Mathematics and Mechanics at the LSU. In 1960, he defended his Ph. D. thesis on algorithmic operators in constructive complete separable metric spaces. His official opponents (examiners) were Vladimir Uspensky and Nikolai Shanin. For several years he continued to study constructive mathematics. Based on new results, partially obtained with his friend Igor Zaslavsky, in 1968 Tseytin defended his second (higher) doctoral thesis [4]. His official opponents were Markov, Boris Trakhtenbrot, and Shanin. In this work, Tseytin summed up his research in the field of constructive mathematics, and never returned to this subject.

Another of Tseytin's major research topics was computational complexity theory. He was a pioneer in the analysis of lower bounds for the time complexity of algorithms and lengths of proofs for propositional formulas. The Markov school worked with Markov normal algorithms, more versatile than Turing machines. (To account for the greater power of Markov algorithms, in 1954, Tseytin enriched the Turing machine appropriately [5].) In 1957, he proved the lower bound $n^2/\log^2 n$ for the time complexity of inverting a word by Markov algorithms. In 1969, Tseytin obtained a tight lower bound $c \cdot n^2$ where the constant c depends on the given inverting algorithm [6].

There are three concepts introduced by Tseytin that are constantly used in the theory of proof complexity.

1. *Tseytin's extension rule* that allows one to introduce new variables and use them to denote arbitrary formulas [3]. This rule makes even a rather weak system, the resolution method, so strong that we still cannot prove exponential lower bounds for the resulting system.

2. *Tseytin tautologies*, systems of linear equations over a finite field constructed according to a given graph. These were the first formulas for which a superpolynomial lower bound for the complexity of propositional proofs was proved. This proof, due to Tseytin, dealt with regular resolutions. This work [3, 9] had a great influence on further studies of the complexity of propositional proofs and became one of the most cited works of the Russian school in mathematical logic and algorithm theory.

3. *Tseytin's efficient translation of propositional formulas into conjunctive normal form*, which became a standard technique in computational complexity theory.

After the completion of his second doctoral thesis, Tseytin's attention switched to computer science. He quickly became the informal leader of the Leningrad programming school. It was most fortunate that a brilliant mathematician became a pioneer and a leader in computer science supporting its autonomy and distinct character. Tseytin pointed out the crucial role of the social dimension in computer science [10]: the collective nature of software creation, the need to adapt the product to human perception, the problem of protecting programs, etc. At that time Tseytin admitted to his colleagues that he no longer considered himself a mathematician.

Tseytin taught numerous courses. A notable example was his course on the theory of algorithms and recursive functions, which was taught over five semesters. His teaching was non-standard. He explained the ideas, tricks and algorithms, but the main emphasis was on complex open problems. He encouraged nontrivial questions, new problems, and new ideas, which he valued more than a simple reproduction of the class material. Students had to put a lot of effort in and between the classes, and many commented that they learned more from Tseytin than from other professors who taught the same course. Here is what some of Tseytin's colleagues had to say:

- Whatever Tseytin did, be it programming or kerosene stove repair, he did it better than anybody else.

- Tseytin's talks for the department were not just interesting, they were always brilliant, a celebration of thought.

Inspired by Noam Chomsky's research on formal grammars, Tseytin became interested in machine processing of texts in natural languages. One facet of this problem was the automatic translation from one language to another. Additional problems emerged that required a deep understanding of the mechanisms underlying natural languages and their description in a form suitable for computer applications. In the 1960s, Tseytin headed a new LSU research group working on experimental systems in this area. Later his group became the Laboratory of Mathematical Linguistics and eventually the Laboratory of Intelligent Systems. At the time, many tried to automate translation from any language to any other. To Tseytin, an avid polyglot and an Esperanto enthusiast, this idea was especially close. Over years, he developed a programming framework based on semantic networks that combined methods of linguistics, formal logic, and systems programming. Automatic translation built on semantic networks has never competed with modern translation tools based on statistical methods. However, Tseytin's methods were capable of extracting the text's meaning and rendering it in applicable areas, such as understanding of natural language, synthesis of computer programs, and artificial intelligence. In the 1980s–90s, using the computers available at the time, the computer scientists working with Tseytin got practical results that were quite advanced even by today's standards.

It was typical of Tseytin to reevaluate his research priorities from the point of view of new results and ideas. In the late 1960s, inspired by the new trends in the theory of algorithmic languages, he focused his research in this field and away from mathematical logic. His 1981 report of this transition titled "From logicism to proceduralism" [8] contains insightful observations about the problems of language processing and artificial intelligence that are still relevant today.

Also at the same time and influenced by the same ideas, a strong group of computer scientists was formed at the Department of Computer Science and the Institute of Mathematics and Mechanics. This group was engaged in the development of compilers for various programming languages. Tseytin was especially interested in Algol-68, a language with a very rich syntax, whose implementation presented a serious scientific challenge. Suffice it to say that only two complete compilers of this language have ever been implemented. Tseytin headed the project at the initial stage and developed the basic ideas; in 1968 the group joined the international cooperation on the development and implementation of this

language [7]. Svyatoslav Lavrov who headed the LSU Dept of Computer Science between 1972 and 1988, an eminent authority in mathematical logic and computer science, wrote that it was under the influence of this remarkable work that the Leningrad school of programming was born and developed. He also noted that even if Algol-68 had not found any application, the emergence of the Tseytin programming school alone would have been an excellent justification of the group existence.

The times were changing, and Tseytin began to cooperate with foreign research and development groups engaged in practical computer science. In 1990–1995, Tseytin worked on making improvements to the MS DOS operating system and creating a text database. Next, in 1995–1996, he worked with the Italian company Microstar Software Ltd.

In 1997, Tseytin spent several months at the IBM Almaden Research Center in California doing research in phenomenal data mining. In the spring of 1999, he continued this work at Trinity College Dublin [11] where he also taught data mining.

Later, in 1999, Tseytin participated in a natural language processing project at the University of New Mexico. This project only lasted one year, after which Tseytin decided to stay in the USA for good. With the support of prominent experts from several countries, he applied for permanent residency under the Extraordinary Ability category. One of those experts was the famous Stanford professor John McCarthy who highly appreciated Tseytin's research, promoted Tseytin's 1997 visit to the Almaden Center, and helped Tseytin during his first years in the USA. Tseytin became a permanent US resident in 2002 and an American citizen in 2008.

In 2000, Tseytin moved to the San Jose, CA, area. He first worked at Rational Software Corporation on their Purify Software, a tool for runtime analysis of complex computer software. In 2003, Rational Software was acquired by IBM, where Tseytin worked with the same group until 2009, receiving 4 patents in the process. After that Tseytin worked for 4 years at Stanford University in the project of the philosopher and logician Patrick Suppes on the construction of an automatic learning system for schoolchildren. Eventually, the project matured, was sold to a commercial company, and now is used as part of Redbird Personalized Learning System.

Tseytin was well into retirement age, but he did not wish to retire and continued to interview for other positions. He did some work with a startup company, was a visiting professor at Stanford, volunteered for political campaigns and the advocacy group Indivisible, and was employed by the Census Bureau for the 2020 population count. Unfortunately he never found in the USA an adequate application for his extraordinary capabilities.

Throughout his life, Tseytin was very active in the scientific community. He was an active member of the Leningrad (later St. Petersburg) Mathematical Society almost from the time of its resumption in 1959. Starting in 1989, he became deeply involved in the work of the St. Petersburg Union of Scientists (SPbSU) serving as a member of its coordinating council. After his departure, the SPbSU commissioned Tseytin to be the union's representative in the USA.

Tseytin had a huge and rare gift. People around him admired his mathematical talent, which was realized in many of his remarkable works. His achievements have been universally recognized by the scientific community. Yet, any sense of superiority was completely foreign to him. People were often surprised with his genuine kindness and sincere will-

ingness to help colleagues in any work. His talent was not only in the generation of new ideas, but also in his perseverance in completing their implementation. It is challenging to match rare abilities with suitable professional opportunities, and unfortunately, there are times when some talents are not adequately appreciated. Tseytin had periods when his abilities were not in demand, or economic conditions impeded his professional progress. We will remember this amazing scientist and a wonderful and generous person who, despite formidable obstacles, managed to bring to life so many of his ideas. See Tseytin's main papers at `http://mathsoc.spb.ru/pers/tseytin/bib.html`

References

[1] "Concerning the problem of recognizing properties of associative calculi," *Doklady Akad. Nauk USSR* 107:2 (1956) 209–212 (in Russian)

[2] "Associative calculi with undecidable equivalence problems," *Doklady Akad. Nauk USSR* 107:3 (1956) 370–371 (in Russian)

[3] "On the complexity of derivation in propositional calculus," in "Studies in constructive mathematics and mathematical logic, part 2" (A.O. Slisenko, editor), Consultants Bureau, New York 1970 115–125

[4] "Research in constructive calculus (constructive real numbers, pointwise specified functions)," Doctor of Sciences thesis. Leningrad State University 1968 (in Russian)

[5] "Reduced form of normal algorithms and a linear acceleration theorem," *J. Math. Sci.* 1 (1973) 148–153

[6] "Lower estimate of the number of steps for an inverting normal algorithm and other similar algorithms," *J. Math. Sci.* 1 (1973) 154–168

[7] "Algol 68. Methods of implementation (G.S. Tseytin, editor), Leningrad State University 1976 224 pages (in Russian)

[8] "From logicism to proceduralism (an autobiographical account)," in *Algorithms in modern mathematics and computer science*, Springer Lecture Notes in Computer Science 122 (1981) 390–396

[9] "On the complexity of derivation in propositional calculus," in "Automation of Reasoning: Classical Papers on Computational Logic 1967–1970," (J.H. Siekmann and G. Wrightson, editors), Springer 1983 466–483

[10] "Is mathematics part of computer science?" *Computer-based tools in education* no. 5 (1999) 3–7 (in Russian)

[11] "Tracing individual public transport customers from an anonymous transaction database" (with M. Hofmann, M. O'Mahony, D. Lyons), *Journal of Public Transportation — University of South Florida* 9:4 (2006) 47–60

Sergei N. Artemov, Lev D. Beklemishev, Leo Borkin, Evgeny Dantsin, Yuri Gurevich, Edward A. Hirsch, Ildar A. Ibragimov, Gene Kalmens, Dmitri Koubenski, Vladik Kreinovich, Andrei A. Lodkin, Yuri V. Matiyasevich, Boris A. Novikov, Vladimir P. Orevkov, Aleksey L. Semenov, Alexander Shen, Anatol Slissenko, Anatoly M. Vershik

SIROCCO Prize for Innovation in Distributed Computing

Laudatio for Boaz Patt-Shamir

It is our pleasure to award the 2023 SIROCCO Prize for Innovation in Distributed Computing to Boaz Patt-Shamir, professor at Tel-Aviv University, Israel, for his outstanding contributions to distributed computing under bandwidth limitations.

Boaz Patt-Shamir is one of the main contributors to distributed network computing, especially regarding solving graph problems efficiently under the constraint that nodes can exchange limited information with their neighbors in each round. Typical problems addressed in Boaz Patt-Shamir's papers are routing, matching, shortest paths, sorting, and checkability, to mention just a few. They also include problems motivated by applications such as sensor networks, recommendation systems, peer-to-peer applications, etc.

At the beginning of the 2000s, Boaz Patt-Shamir co-authored two breakthrough papers, namely:

- Zvi Lotker, Boaz Patt-Shamir, David Peleg: Distributed MST for constant diameter graphs. In Proc. of *20th ACM Symposium on Principles of Distributed Computing* (PODC 2001).

- Zvi Lotker, Elan Pavlov, Boaz Patt-Shamir, David Peleg: MST construction in $O(\log \log n)$ communication rounds. In Proc. of *15th ACM Symposium on Parallelism in Algorithms and Architectures* (SPAA 2003).

Indeed, since the work by J. Garay, S. Kutten, D. Peleg, and V. Rubinovich (SIAM J. of Computing, 1998 and 2000), it was known that constructing a minimum-weight spanning tree (MST) can be done in roughly $O(D + \sqrt{n})$ rounds in n-node networks with diameter $D = \Omega(\log n)$ when the nodes are restricted to exchange $O(\log n)$-bit messages in each round. It was also known that this bound is essentially tight. In his PODC 2001 paper, Boaz Patt-Shamir et al. proved that an $n^{\Omega(1)}$

lower bound also holds in networks with constant diameter, specifically $\tilde{\Omega}(\sqrt{n})$ rounds in networks with diameter 4, and $\tilde{\Omega}(n^{1/4})$ rounds in networks with diameter 3. In contrast, the same paper shows that, in graphs with diameter 2, MST can be solved in $O(\log n)$ rounds, hence establishing an exponential gap between the computing power of networks with diameter 3 and those with diameter 2. Another exponential gap was established in the SPAA 2003 paper, in which Boaz Patt-Shamir et al. proved that MST can be solved in just $O(\log \log n)$ rounds in cliques. The model introduced in the SPAA 2003 paper is now commonly referred to as the *Congested Clique* model, which completes the trio of core models for distributed network computing, now known as LOCAL, CONGEST, and Congested Clique.

The Congested Clique model enables to focus solely on the impact of congestion caused by narrow communication links, putting aside effects caused by the distance between nodes, and, more generally, by the structure of the network. As such, the Congested Clique model is an elegant, flexible, and fruitful abstraction enabling to understand the power and limitation of computing with limited communication resources. Unsurprisingly, the SPAA paper introducing the Congested Clique model had a huge impact on research in the framework of principles of distributed computing. Last but not least, the Congested Clique model links together distributed and parallel computing, thanks to its close connections to the Massively Parallel Computation (MPC) model, which emerged in the years 2008–2012 owing to the deployment of popular computing platforms such as MapReduce and Hadoop.

For his participation to the discovery of the Congested Clique model, and for all his contributions to distributed computing in models constrained by bandwidth limitations such as Congested Clique, CONGEST, sensor networks, etc., Boaz Patt-Shamir fully deserves to be recognized by the distributed computing community as one of its prominent members.

The 2023 Award committee:[1]

Paola Flocchini (University of Ottawa, Canada)
Magnús M. Halldórsson (Reykjavik University, Iceland)
Tomasz Jurdziński (University of Wroclaw, Poland)
Zvi Lotker (Ben-Gurion University of the Negev, Israel)
Merav Parter (Weizmann Institute, Israel)
Andréa W. Richa (Arizona State University, USA)
Stefan Schmid (Technical University of Berlin, Germany)

[1] We wish to thank the nominators for the nomination and for contributing heavily to this text.

ICALP 2024

51st EATCS International Colloquium on Automata, Languages and Programming
Tallinn, Estonia, July 8-12, 2024
https://compose.ioc.ee/icalp2024/

Call for Papers

The 51st EATCS International Colloquium on Automata, Languages, and Programming (ICALP) will take place in:

Tallinn, Estonia, July 8-12, 2024

ICALP is the main conference and annual meeting of the European Association for Theoretical Computer Science (EATCS). As usual, ICALP will be preceded by a series of workshops, which will take place on July 7.

The 2024 edition has the following features:

- Submissions are anonymous and there is a rebuttal phase. - The conference is planned as a physical, in-person event. - ICALP 2024 is co-located with Logic in Computer Science (LICS) 2024 and Formal Structures for Computation and Deduction (FSCD) 2024.

Important dates

Submissions: February 14, 2024 (1pm CET)
Rebuttal: March 26-29, 2024
Author notification: April 14, 2024
Camera-ready version: April 28, 2024
Early registration: TBA
Conference: July 8-12, 2024 (Workshops on July 7)

Deadlines are firm; late submissions will not be considered.

Conference website: https://compose.ioc.ee/icalp2024/

Submission guidelines:

1) Papers must present original research on the theory of computer science. No prior publication and no simultaneous submission to other publication outlets (either a conference or a journal) is allowed. Authors are encouraged to also make full versions of their submissions freely accessible in an on-line repository such as ArXiv, HAL, ECCC.

2) Submissions take the form of an extended abstract of no more than 15 pages, excluding references and a clearly labelled appendix. The appendix may consist either of omitted proofs or of a full version of the submission, and it will be read at the discretion of program committee members. The use of the LIPIcs document class is an option, but not required. The extended abstract has to present the merits of the paper and its main contributions clearly, and describe the key concepts and technical ideas used to obtain the results. Submissions must provide the proofs which can enable the main mathematical claims of the paper to be verified.

3) Submissions are anonymous. The conference will employ a lightweight double-blind reviewing process. Submissions should not reveal the identity of the authors in any way. Authors should ensure that any references to their own related work are in the third person (e.g., not "We build on our previous work ..." but rather "We build on the work of ..."). The purpose of this double-blind process is to help PC members and external reviewers come to an initial judgment about the paper without bias, and not to make it impossible for them to discover who the authors are if they were to try. Nothing should be done in the name of anonymity that weakens the submission or makes the job of reviewing the paper more difficult. In particular, important references should not be omitted. In addition, authors should feel free to disseminate their ideas or draft versions of their paper as they normally would. For example, authors may post drafts of their papers on the web, submit them to arXiv, and give talks on their research ideas.

4) Submissions authored or co-authored by members of the program committee are allowed.

5) The submissions are done via Easychair to the appropriate track of the conference (see topics below). The use of pdflatex or similar pdf generating tools is mandatory and the page limit is strict (see point 2.) Papers that deviate significantly from these requirements risk rejection without consideration of merit.

6) During the rebuttal phase, authors will have from March 26-29, 2024 to view and respond to initial reviews. Further instructions will be sent to authors of submitted papers before that time.

7) At least one author of each accepted paper is expected to register for the conference, and all talks are in-person. In exceptional cases, there may be support for

remotely presenting a talk.

8) Papers authored only by students should be marked as such upon submission in order to be eligible for the best student paper awards of the track.

Awards

During the conference, the following awards will be delivered:

– the EATCS award,
– the Gödel prize,
– the Presburger award,
– the EATCS distinguished dissertation award,
– the best papers for Track A and Track B,
– the best student papers for Track A and Track B.

Proceedings

ICALP proceedings are published in the Leibniz International Proceedings in Informatics (LIPIcs) series. This is a series of high-quality conference proceedings across all fields in informatics established in cooperation with Schloss Dagstuhl – Leibniz Center for Informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge. The accepted papers will need to comply with the LIPIcs style.

Topics

Papers presenting original research on all aspects of theoretical computer science are sought. Typical but not exclusive topics of interest are:

Track A: Algorithms, Complexity and Games

- Algorithmic and Complexity Aspects of Network Economics
- Algorithmic Aspects of Biological and Physical Systems
- Algorithmic Aspects of Networks and Networking
- Algorithmic Aspects of Security and Privacy
- Algorithmic Game Theory and Mechanism Design
- Approximation and Online Algorithms
- Combinatorial Optimization

- Combinatorics in Computer Science
- Computational Complexity
- Computational Geometry
- Computational Learning Theory
- Cryptography
- Data Structures
- Design and Analysis of Algorithms
- Distributed and Mobile Computing
- Foundations of Machine Learning
- Graph Mining and Network Analysis
- Parallel and External Memory Computing
- Parameterized Complexity
- Quantum Computing
- Randomness in Computation
- Sublinear Time and Streaming Algorithms
- Theoretical Foundations of Algorithmic Fairness

Track B: Automata, Logic, Semantics, and Theory of Programming

- Algebraic and Categorical Models of Computation
- Automata, Logic, and Games
- Database Theory, Constraint Satisfaction Problems, and Finite Model Theory
- Formal and Logical Aspects of Learning
- Formal and Logical Aspects of Security and Privacy
- Logic in Computer Science and Theorem Proving
- Models of Computation: Complexity and Computability

- Models of Concurrent, Distributed, and Mobile Systems
- Models of Reactive, Hybrid, and Stochastic Systems
- Principles and Semantics of Programming Languages
- Program Analysis, Verification, and Synthesis
- Type Systems and Typed Calculi

ICALP 2024 Programme Committee

Track A: Algorithms, Complexity, and Games

Nima Anari (Stanford University)
Karl Bringmann (co-chair, Saarland University)
Parinya Chalermsook (Aalto University)
Vincent Cohen-Addad (Google Research)
Jose Correa (Universidad de Chile)
Holger Dell (Goethe University Frankfurt)
Ilias Diakonikolas (University of Wisconsin-Madison)
Yuval Filmus (Technion)
Arnold Filtser (Bar Ilan University)
Naveen Garg (IIT Delhi)
Pawel Gawrychowski (University of Wrocław)
Anupam Gupta (Carnegie Mellon University)
Samuel Hopkins (MIT)
Sophie Huiberts (Columbia University)
Giuseppe Italiano (LUISS University)
Michael Kapralov (EPFL)
Eun Jung Kim (Université Paris-Dauphine)
Sándor Kisfaludi-Bak (Aalto University)
Tomasz Kociumaka (Max-Planck-Institute for Informatics)
Fabian Kuhn (University of Freiburg)
Amit Kumar (IIT Delhi)
William Kuszmaul (Harvard University)
Rasmus Kyng (ETH Zurich)
Kasper Green Larsen (Aarhus University)
François Le Gall (Nagoya University)
Pasin Manurangsi (Google Research)
Daniel Marx (CISPA Helmholtz Center for Information Security)

Yannic Maus (TU Graz)
Nicole Megow (University of Bremen)
Ruta Mehta (University of Illinois at Urbana-Champaign)
Jakob Nordström (University of Copenhagen)
Richard Peng (University of Waterloo)
Seth Pettie (University of Michigan)
Adam Polak (Bocconi University)
Lars Rohwedder (Maastricht University)
Eva Rotenberg (DTU Compute)
Sushant Sachdeva (University of Toronto)
Melanie Schmidt (University of Cologne)
Sebastian Siebertz (University of Bremen)
Shay Solomon (Tel Aviv University)
Nick Spooner (University of Warwick)
Clifford Stein (Columbia University)
Ola Svensson (co-chair, EPFL)
Luca Trevisan (Bocconi University)
Ali Vakilian (Toyota Technological Institute Chicago)
Jan van den Brand (Georgia Tech)
Erik Jan van Leeuwen (Utrecht University)
Oren Weimann (University of Haifa)
Nicole Wein (University of Michigan)
Andreas Wiese (TU Munich)
John Wright (UC Berkeley)

Track B: Automata, Logic, Semantics, and Theory of Programming

Arnold Beckmann (Swansea University)
Manuel Bodirsky (TU Dresden)
Patricia Bouyer (LMF Cachan)
Yijia Chen (Shanghai Jiao Tong University)
Victor Dalmau (Universitat Pompeu Fabra)
Laurent Doyen (CNRS, LMF)
Marcelo Fiore (Cambridge University)
Stefan Göller (University of Kassel)
Martin Grohe (RWTH Aachen University, chair)
Sandra Kiefer (Oxford University)
Aleks Kissinger (Oxford University)
Bartek Klin (Oxford University)

Antonin Kucera (Masaryk University Brno)
Carsten Lutz (University of Leipzig)
Jerzy Marcinkowski (University of Wrocław)
Annabelle McIver (Macquaire University Sidney)
Andrzej Murawski (Oxford University)
Pawel Parys (University of Warsaw)
Michał Pilipczuk (University of Warsaw)
Joel Ouaknine (Max Planck Institute for Software Systems)
Christian Riveros (Pontificia Universidad Catolica de Chile)
Alexandra Silva (Cornell University)
Balder ten Cate (ILLC Amsterdam)
Szymon Toruńczyk (University of Warsaw)
Igor Walukiewicz (CNRS, University of Bordeaux)
Sarah Winter (IRIF, University Paris Cité)
Georg Zetzsche (Max Planck Institute for Software Systems)
Martin Ziegler (KAIST)

ICALP 2024 Workshops
The call and the selection of workshops will be done jointly with LICS. The first call will be issued in October.

ICALP 2024 Proceedings Chairs
Gabriele Puppis (University of Udine, Italy)

ICALP-LICS-FSCD 2024 Organizing Committee

Pawel Sobocinski (Tallinn University of Technology) Conference Chair
Niccolò Veltri (Tallinn University of Technology)
Amar Hadzihasanovic (Tallinn University of Technology)
Fosco Loregian (Tallinn University of Technology)
Matt Earnshaw (Tallinn University of Technology)
Diana Kessler (Tallinn University of Technology)
Kristi Ainen (Tallinn University of Technology)

Institutional Sponsors

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

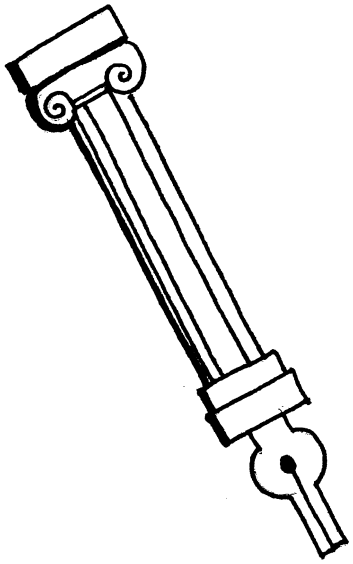
CWI, Centum Wiskunde & Informatica
Amsterdam, The Netherlands

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: Michał Pilipczuk

Bio: *Michał Pilipczuk is an associate professor at the Institute of Informatics of the University of Warsaw, Poland, to where he returned after earning his PhD degree at the University of Bergen, Norway. His research mostly revolves around structural graph theory, its applications in the design of algorithms (most often parameterized), and connections with logic. He received several awards, including the Witold Lipski Prize in 2015 and ERCIM Cor Baayen Award in 2016, and was a recipient of an ERC Starting Grant in 2020.*

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?

MiPi: Let me actually provide two photos, which show me in my two natural habitats. The first one is from my office in Warsaw; the scribbles in the background are some actual math. The second one is from a week-long hike we did this year in Apuseni mountains in Romania together with my friends (and coauthors) Karolina Okrasa and Filip Mazowiecki. We try to find time for such holidays in the wilderness every year.

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

MiPi: Similarly to many of my Polish colleagues, my adventure with mathematics started for real in high school with participation in the Polish Mathematical Olympiad. Later I became heavily involved in the organization of the olympiad, including chairing the problem selection committee for some time. For a few years I also served as the leader of the Polish team at the International Mathematical Olympiad. Even though I drifted away from these duties a few years ago, outreach activities for talented high school students are still very close to my heart. So you can meet me once in a while at various mathematical camps, where I have the pleasure of teaching interesting mathematics to young and fresh minds.

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

MiPi: Personally I find it extremely hard to motivate myself to read a paper, I much more prefer to see it explained at a blackboard. Nevertheless, it was quite often the case that when I eventually forced myself to do it, it paid off really well, because in this way you learn tricks that lie at the very bottom of the reasoning. A particular example that comes to my mind is the paper *Structure Theorem and Isomorphism Test for Graphs with Excluded Topological Subgraphs*, by Martin Grohe and Dániel Marx (SIAM J. Computing, 2015). I believe this paper taught me how to design and prove decomposition theorems for graphs.

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

MiPi: I am not sure if I can recommend any single paper. Most of them are probably quite boring reads, which in the hindsight should have been presented differently. But one material I can definitely suggest, and of which I am quite proud, are the lecture notes on the theory of sparse graphs that we wrote together with Sebastian Siebertz and Marcin Pilipczuk. They are freely available on the website of the second edition of our Warsaw course on this topic: `https://www.mimuw.edu.pl/~mp248287/sparsity2/`.

The story is that Sebastian came to Warsaw in 2016 for a two-years POLONEZ fellowship[1]. I was his research partner and the goal was to work on the said theory of sparse graphs. At this point we were both very interested in this topic: it was clear that the area had a huge potential and was on the brink of fundamental developments. To get a better grasp, in 2017 we decided to run together a course about this subject at the University of Warsaw, and simultaneously write high-quality lecture notes. This was an excellent decision: in this way we forced ourselves to deeply understand which arguments and notions are truly fundamental, and how they should be presented. This understanding had a tremendous impact on our research, and led to multiple discoveries.

The lecture notes got updated when we repeated the course with my brother Marcin two years later. We also supplemented them with recordings of all the lectures; these recordings are freely available on youtube. The plan is to eventually turn the lecture notes into a textbook, but this project is unfortunately stalled at the moment. And we definitely should get back to it, as the theory of sparse graphs is a beautiful piece of graph theory with a wealth of powerful tools that should find a much wider recognition in the broad TCS community.

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

MiPi: When it comes to creative work, for instance trying to come up with an idea on how to attack a problem, I very much enjoy thinking alone while walking outdoors. So a longer Saturday hike in a forest would be my preferred way of approaching a harder scientific nut. For less brain-demanding work, for instance writing up papers or working on reviews, I like to work in trains, somehow this helps with concentration. Though, calmer evening hours at home are also good for this purpose.

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

MiPi: Patience and persistence are not my strong sides when it comes to research... If after a week or two a problem does not give in, I typically like to put it

[1]POLONEZ is a clone of the Marie Skłodowska-Curie individual fellowship programme, it offers fellowships at Polish academic institutions. It is run by the Polish National Science Center (NCN) and is funded from ERC funds.

on a shelf to mature. Perhaps after a while some new ideas will emerge. Indeed, we had quite a few good problems that "cracked" after a few years of maturing in this way. I also like to juggle with several projects at the same time with different research groups, so that one can focus on directions that seem interesting or promising at the current moment.

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

MiPi: As some readers may know, I have an older brother Marcin, who also works in our algorithmic group at the University of Warsaw. In fact, we work together a lot; probably at least a third of my papers are coauthored with him. Having a brother with the same initials (Thanks, Mom and Dad!) working in the same area can lead to multiple confusing situations. At least once a month one of us receives some semi-sensitive e-mail addressed to the other; it is unclear how many reviews each of us did that were actually meant for the other one. At some point there was alarm in ERC administration when they figured out that there was an M. Pilipczuk in Warsaw that was holding two ERC grants at the same time. My personal favourite was when my brother was called to testify in a court case, and he needed to clarify to the court right at the start of his testimony that they invited the wrong M. Pilipczuk.

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

MiPi: I think my only piece of advice would be not to take advice of senior people too seriously, particularly of some supposed experts in various bulletins. A bit more seriously, relying on external opinions, especially when making career decisions, should be done with caution. When it comes to work-life balance, advice from more experienced colleagues may very often be biased in the work direction, because they will never see the full picture of your goals, motivations, and feelings outside of the work context. It is good to ask some friends you trust for their thoughts, just not to miss any obvious points. But make sure that at the end of the day, you always make the decision yourself.

And about investing time, just invest time in your private life. If you like what you do, papers will come about anyways.

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

MiPi: I guess "passion" is a heavily overused word, but you just need to truly enjoy thinking about mathematical problems. If you have it, then it is a dream job for you, and otherwise it will not work out.

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

MiPi: I believe theoretical computer science has now matured as an area of mathematics, but maintained a lot of unhealthy habits from its puberty. This mostly concerns publishing practices and research culture. For instance, when TCS was young, papers were a dozen pages long and could be read during a reasonably long bus ride. Nowadays, a typical paper at FOCS, STOC, LICS, or SODA contains 50 pages of highly nontrivial mathematics. Yet, we still pretend that skimming through the first 10 pages and giving an educated guess on whether the results are interesting is a good enough "peer-review". Even worse, both FOCS and SODA have recently lifted the page limit on camera ready versions. This means that now everyone can just publish all the technical sections that literally nobody read, which, from the point of view of scientific integrity, is just an acquiescence to systemic laziness. I consider these developments distinctly negative.

I believe this issue is a symptom of a larger problem that in theoretical computer science, particularly in algorithms, the increasingly dominant understanding of research is that of a race to obtain a yet another improved result, and consequently a yet another paper. And in a race there is little time to look back, or even to contemplate the goal of the race. The best example of this is the phrase "we are the first to...", which slowly becomes a must-have in every abstract of an algorithmic paper. The classic mathematical culture of humility is slowly being replaced by a culture of bragging. Admittedly, all of this is partly fueled by the overblown expectations about publication records in the academic job market, but partly it is also a consequence of how theoretical computer science developed as a field.

I do not really have any concrete proposition on how to address these issues, but my feeling is that the current situation and the direction of changes are detrimental in the long run. We should, as a community, think consciously on how to implement more scientifically mature practices. Transitioning from the conference-based publication culture to a journal-based one, or some kind of a hybrid model, would be a natural step.

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

MiPi: Outside of mathematics I am also very interested in history, particularly contemporary, say of 20th century. Nowadays in the internet one can find a wealth of great history-related content. One particular pointer that I would like to recommend is a class on the history of Ukraine given at Yale University by Timothy Snyder. The course consists of 23 lectures, all of which are freely available on youtube. I was impressed by the extent of the material presented, how all the intricate connections between different actors in the past have been explained in order to give a really deep understanding of the background and motives behind

the current Russo-Ukrainian war. This is a great piece of content that can provide a western viewer a lot of much needed context of what is happening now.

Please complete the following sentences?

- *Being a researcher...* was the obvious choice for me.
- *My first research discovery...* still did not make it to any journal :(
- Being a happy human being ... *is key to being a happy academic.*
- *Theoretical computer science in 100 years from now...* will still have no clue about the P vs NP question.

The Viewpoint Column

BY

Stefan Schmid

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

BEATCS no 141

Removing the Barriers: Overcoming Impostor Phenomenon as a Community

Ulrike Schmidt-Kraepelin

Abstract

The *impostor phenomenon* (aka *impostor syndrome*) is a wide-spread problem in academia, especially in fields that (supposedly) require "natural talent" or "genius". The phenomenon is particularly prevalent among underrepresented groups and early-career researchers. Overcoming feelings of self-doubt and perceived inadequacy is often left to the individual, which exerts a heavy mental load and a competitive disadvantage. In this article, I argue that any efforts to make our research community more diverse should especially aim to mitigate impostor phenomenon for all current and future members. To this end, I offer concrete suggestions for community members wishing to contribute to this endeavor.

As for many young researchers, self-doubt was a constant companion throughout my PhD. Now that I am coming to grips with these concerns myself, I keep wondering about the broader impact of self-doubt on our field. If I experienced these feelings despite many privileges and a very supportive environment, how does self-doubt impact those less privileged? Why is self-doubt so concentrated in certain demographic groups and academia in general? What can we, as a research community, do for future researchers? In this article, I want to share some of my (partial) answers to these questions. In the first part, I summarize recent literature and argue why the community should take more responsibility. In the second part, I derive concrete suggestions for change.

The Impostor Phenomenon, Its Impact, and our Responsibility

The *impostor phenomenon*, first described by Clance and Imes [1978], is a psychological phenomenon where individuals, despite external evidence of their competence and accomplishments, believe they do not deserve the success they have achieved. A person experiencing impostor phenomenon (Slank [2019] calls them "IPP") feels like they are just pretending to be competent. As a consequence, they fear that others will eventually discover their supposed inadequacies. IPPs attribute their accomplishments to luck or external factors and their failures to

their assumed lack of competence. This phenomenon is also frequently referred to as the "impostor syndrome". However, in line with the psychology literature [Clance and Imes, 1978], I prefer the term "impostor phenomenon", since it avoids the connotation of an individuals' psychological deficit. In contrast to this connotation, I will argue below for an understanding of the phenomenon as a natural reaction to certain environments and biases faced by people from underrepresented groups. Similar arguments have been put forward, e.g., by Olah [2019] and Tulshyan and Burey [2021].

Effects and consequences. Individuals experiencing impostor phenomenon witness increased fear of failure and psychological distress. For example, this may include anxiety about exams, presentations, but also casual research conversations. IPPs are less likely to ask questions, be proactive, and expand their professional network. Impostor phenomenon has also been related to a decreased sense of *belonging* [Muradoglu et al., 2022], i.e., feeling connected to others, which is a basic need that is closely connected to motivation, interest, and persistence. IPPs also appear to pursue qualitatively different goals: Kumar and Jagacinski [2006] found that IPPs have a higher tendency to pursue *performance goals*, i.e., they derive feelings of competence from outperforming others or avoiding failure compared to others. In contrast, individuals not experiencing impostor phenomenon have a higher tendency to purse *task goals*, i.e., they focus on learning and understanding the task; for them task mastery is motivated intrinsically. Even without considering the negative psychological effects resulting from pursuing performance goals, it seems evident that pursuing task goals is much more effective for building a career in theoretical computer science and also for the progress of our field as a whole. Lastly, impostor phenomenon has been found to be strongly correlated with perfectionism [Henning et al., 1998]. Perfectionist behavior, which is pursued in order to make up for the perceived lack of ability, can lead to overwork and prioritization issues. Ironically, pairing perfectionism with a performance mindset can fuel the impostor phenomenon even more, as spending more time than others on a task is interpreted as proof of intellectual inferiority. All in all, impostor phenomenon decreases an early-career researcher's quality of life and sets hurdles in the way of accessing their full academic potential. In particular, impostor phenomenon thus quite directly limits the scientific progress of our field.

Impostor phenomenon in academia, brilliance, and stereotypes. Since the 1970s, the impostor phenomenon has been extensively studied in psychology. Key findings are that the risk of experiencing impostor phenomenon varies by demographic group, amount of experience in a field, and work environment. In general, impostor phenomenon has been found to be prevalent in academia, es-

pecially among early-career researchers. In a survey of over 4000 academics, Muradoglu et al. [2022] moreover identified a strong correlation between the occurrence of impostor phenomenon and the amount to which a research field values "innate talent" or "brilliance". Clearly, theoretical computer science falls into this category [Leslie et al., 2015]. Even more concerning, the authors found that underrepresented groups in these fields (such as women and some ethnic groups) are significantly more likely to experience impostor phenomenon, and that this disparity grows as a function of the extent to which the research field is brilliance-oriented. Interestingly, brilliance-orientation is also closely related to the number of researchers from these groups. Leslie et al. [2015] uncovered this correlation in their seminal study and conclude that "the extent to which practitioners of a discipline believe that success depends on sheer brilliance is a strong predictor of women's and African American's representation in that discipline". Hence, brilliance-orientation and impostor phenomenon can be hypothesized to discourage underrepresented groups from joining or staying in academia. Leslie et al. [2015] explain this effect with stereotypes of women and some ethnic groups possessing less innate talent. Such stereotypes not only lead to biases of evaluators, but also make these groups prone to stereotype threats[1] and self-selection biases. Sadly, these stereotypes appear to persist. For example, Napp and Breda [2022] recently confirmed that girls are still stereotyped to possess less innate talent – paradoxically – even more so in gender-egalitarian countries.

Responsibility. If the community leaves the responsibility of overcoming impostor phenomenon to the individual, this induces an additional burden and a competitive disadvantage that might be too large to compensate for in some cases. Hence, it is evident that interventions have to take place. The research summarized above indicates that the prevalence of impostor phenomenon among certain demographic groups is not a fact of life but a reaction to persisting stereotypes about what is thought to be one of the crucial prerequisites for success in their fields, i.e., "natural talent". Yet, the most common interventions against impostor phenomenon are targeted at the underrepresented groups rather than their environment. These initiatives serve an important purpose, as they can in particular increase the sense of belonging. Nevertheless, I believe that restricting ourselves to these group-targeted initiatives would be problematic for multiple reasons: Targeted groups can get the impression that they have a "condition" that needs to be fixed, whereas we should in fact fix the environment. Second, as previously stressed, all early-career researchers in our field are at increased risk of experiencing impostor phenomenon, albeit not equally. Hence, any strategy that solely

[1] Stereotype threat describes the well-studied theory that negative stereotypes can decrease the performance of individuals, even without the individual needing to subscribe to the stereotype.

targets specific demographic groups is at danger of overlooking early-career researchers that come from groups that are targeted less often. For example, first-generation students are at increased risk of experiencing impostor phenomenon, nevertheless they are rarely targeted by initiatives. Last, the traditional approach does not promise to be sustainable if it has to address each new generation of junior researchers, without improving the underlying conditions. Complementing existing interventions, I therefore want to discuss actions that we all can take, which might reduce impostor phenomenon in early-career researchers in the first place. This allows the community to take responsibility for a problem that is not caused by an inadequacy of affected groups, but rather by our stereotype-prone society. I am not alone in this assessment, as Muradoglu et al. [2022] conclude that "brilliance-oriented fields have failed to create an environment in which women, particularly those from groups underrepresented in academia, and early-career academics feel capable of succeeding. Thus, the onus of reducing impostor feelings should be on the fields, not on the academics themselves."

What Can We Do? Overcoming Impostor Phenomenon as a Community

Hopefully, the preceding discussion has convinced the reader that overcoming impostor phenomenon is in the shared interest of all members of the theoretical computer science community. Not only does impostor phenomenon have a negative impact on life quality and mental health of many of our colleagues, but it also inhibits the development of many early-career researchers, reduces scientific progress, and hinders efforts of increasing the field's diversity. In the following, I suggest actions we can take as educators, advisors, and colleagues that can mitigate impostor phenomenon in students and researchers. I am not claiming that this list is exhaustive, nor that it should be followed blindly. Rather, I aim to initiate (and support ongoing) discussions. I derived some of the suggestions from reflecting on the literature discussed above; others stem from personal experience of colleagues and myself.

1. **De-emphasizing brilliance and innate talent:** The question to which extent innate talent is necessary to succeed in theoretical computer science is up for debate and certainly out of scope for this article. However, we can at least acknowledge that (over)emphasizing the role of talent can work against the aim to diversify our research field and carries the risk of worsening the situation for groups that are negatively stereotyped to have less innate talent. Leslie et al. [2015] even go one step further by concluding that "[their] data suggest that academics who wish to diversify their fields might want to downplay talk of innate intellectual giftedness and instead highlight the importance of sustained effort for top-level success in their field."

2. **Creating an environment of growth:** Individuals who view intelligence as a fixed entity which cannot be changed are more likely to experience impostor fears (see, e.g., [Kumar and Jagacinski, 2006]). Luckily, this "entity theory" has been frequently challenged, e.g., by Dweck [2006] who offers the idea of a *growth mindset*. In a nutshell, the idea is that abilities are developed and that learning, challenges, and setbacks should be embraced as a source of growth. I believe that it is vital that research groups and collaborators communicate the core ideas of a growth mindset. For example, this can mitigate the issue that individuals experiencing impostor phenomenon are often hesitant to ask questions, as they assume that every other person in the room knows the answer and their question will expose their assumed intellectual inferiority. In contrast, a "growth environment" would not judge questions but rather welcome them in order to jointly derive a deep understanding of the problem at hand.

3. **Giving constructive and specific feedback:** One might be tempted to think that individuals experiencing impostor phenomenon would just need enough praise and cannot deal with criticism. I strongly disagree and believe that advisors should empower students to objectively assess their own work. Specifically, mindless praise might make an IPP believe that they "fooled yet another person." On the other hand, specific and honest positive comments about their work are easier to accept and more helpful. Similarly, constructive feedback coming from a growth mindset is extremely powerful. Ideally, (if true), the advisor can communicate that they believe in the students ability to succeed in the field despite seeing areas for improvement, and that their feedback aims to support the scientific development of the student. Moreover, I believe it is important to introduce students to techniques for receiving and giving constructive feedback early on. I learned about such techniques within a masters' seminar and found them extremely helpful later on.

4. **Detecting perfectionism:** Many individuals experiencing impostor phenomenon tend to overcompensate by spending enormous time on perfecting their work (e.g., posters and recorded talks). In the short run, this behavior rarely appears to be disadvantageous, it can provide a feeling of security, and it is often even encouraged. However, if students struggle prioritizing, this behavior can lead to overworking, inefficient allocation of efforts, and high opportunity costs. Here, I think that advisors and collaborators can play a crucial role in communicating clear expectations and giving feedback on what is "good enough." Personally, I do remember the relief I felt when my PhD advisor protected me from my own overly high expectations

regarding unimportant tasks by simply saying "let's try to finish this today, it really doesn't have to be perfect."

5. **Creating awareness:** Discussing the impostor phenomenon and its impact with students at an early stage serves at least two purposes: (i) Those students experiencing impostor phenomenon understand that these feelings are – unfortunately – rather common and should not be interpreted as a signal regarding their abilities but as a reaction towards their environment. Also, hearing that even successful researchers have experienced similar feelings at some point in their careers (even if not labeled as "impostor phenomenon") can be empowering. (ii) Those students that do not experience impostor phenomenon can help to improve the situation for their fellow students by acting more empathically and not interpreting others' insecurities as a lack of competence. Personally, I remember a discussion with a fellow PhD student who had never heard of impostor phenomenon before and who could not imagine why someone would be hesitant to ask a question in front of an audience.

6. **Encouraging applications:** Clearly, students and early-career researchers who doubt their own abilities are less likely to apply for scholarships, awards, selective workshops, or prestigious jobs. By the Matthew effect[2], this can have a long lasting negative effect on their careers. Here, advisors, mentors, and colleagues can make a vital difference by nominating the student or proactively pointing towards calls and emphasizing that they are confident about the suitability of the student. Related to this, decision makers should keep this effect in mind when evaluating academic profiles.

7. **Normalizing failure:** Whereas all academics encounter failures, setbacks, and rejections, individuals experiencing impostor phenomenon might interpret them as "proof" for their lack of competence. Thus, it is important for more established researchers to share their failures, in particular with early-career researchers. A prominent example are "CVs of failure"[3], but also occasional examples in casual conversations can be very helpful.

8. **Avoiding judgmental comments:** Researchers often have different and strong opinions about the quality of journals and conference proceedings, or even entire subfields, and discussions about these are popular. When

[2]The Matthew effect describes a hypothesis stating that small initial advantages can build up disproportionately over the course of a career, also summarized by the phrase "the rich get richer".

[3]See, for example https://www.uni-goettingen.de/de/document/download/bed2706fd34e29822004dbe29cd00bb5.pdf/Johannes_Haushofer_CV_of_Failures[1].pd or http://everydayscientist.com/CV/sjl_CV-failures.pdf.

engaging in such discussions, we should be careful with judgmental comments, especially when early-career researchers are around. First, by doing so we might implicitly disparage other researchers work, for example, if they publish at the discussed venues, or even worse, their work got rejected from these venues. Second, the perception of a journal or conference can vary enormously across subfields. Regardless of the reason, disparaging comments can make early-career researchers doubt their accomplishments. Instead, we could emphasize that despite the fact that publication venues are often used as a proxy for the quality of a work, they are by no means a perfect indicator for such.

9. **Avoiding stereotypes:** There are many stereotypes about theoretical computer scientists. (ChatGPT summarizes us as "introverted", "nerdy", "unrelatable", and "male".) While joking about these stereotypes can be empowering for some people, overemphasizing these characteristics can decrease the sense of belonging in individuals that do not match this narrow image.

10. **Reflecting on discussions about affirmative actions:** Several times, some of my fellow students confronted me with the preconception that I was given higher chances of being selected for scholarships or jobs because I was a woman. This is a tricky point, as there is no denial that, in some situations, affirmative action does take place. While I think it is important to maintain an open discussion about the necessity and implementation of these actions, these discussions can also quickly fuel impostor phenomenon among underrepresented groups. It is therefore essential that we reflect on the context and our own intentions before engaging in these discussions. Lastly, we should keep in mind that, even though individuals of underrepresented groups might be prioritized at a handful of moments during their careers, there are good arguments for doing so. Just to name two: As exemplified in this article, it probably took them significantly more struggles to reach this point in their career compared to their peers. Moreover, the visibility and representation they create can pave the way for generations to come, who will hopefully experience less impostor phenomenon.

Acknowledgments. While writing this article, I was faced with my very own set of insecurities: Am I the right person to write this article? Is the issue even big enough? Aren't there enough articles on the topic? Do I want to be associated with the issue? I am extremely grateful to my friends and colleagues who strongly encouraged me to write this article, contributed some of the ideas mentioned above, and provided valuable feedback on a first draft. You know who you are. Thank you for your support and openness.

References

P. R. Clance and S. A. Imes. The imposter phenomenon in high achieving women: Dynamics and therapeutic intervention. *Psychotherapy: Theory, research & practice*, 15(3):241, 1978.

C. S. Dweck. *Mindset: The new psychology of success*. Random house, 2006.

K. Henning, S. Ey, and D. Shaw. Perfectionism, the impostor phenomenon and psychological adjustment in medical, dental, nursing and pharmacy students. *Medical education*, 32(5):456–464, 1998.

S. Kumar and C. M. Jagacinski. Imposters have goals too: The imposter phenomenon and its relationship to achievement goal theory. *Personality and Individual differences*, 40(1):147–157, 2006.

S.-J. Leslie, A. Cimpian, M. Meyer, and E. Freeland. Expectations of brilliance underlie gender distributions across academic disciplines. *Science*, 347(6219): 262–265, 2015.

M. Muradoglu, Z. Horne, M. D. Hammond, S.-J. Leslie, and A. Cimpian. Women—particularly underrepresented minority women—and early-career academics feel like impostors in fields that value brilliance. *Journal of Educational Psychology*, 114(5):1086, 2022.

C. Napp and T. Breda. The stereotype that girls lack talent: A worldwide investigation. *Science Advances*, 8(10):eabm3689, 2022.

N. Olah. "Impostor syndrome" is a pseudo-medical name for a class problem. *The Guardian*, 19, 2019.

S. Slank. Rethinking the imposter phenomenon. *Ethical Theory and Moral Practice*, 22(1):205–218, 2019.

R. Tulshyan and J.-A. Burey. Stop telling women they have imposter syndrome. *Harvard Business Review*, 11, 2021.

THE THEORY BLOGS COLUMN

BY

LUCA TREVISAN

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

Bill Gasarch is a professor of computer science at the University of Maryland at College Park. He works on computational complexity and combinatorics, and he is interested in math education. Bill writes with Lance Fortnow (who was featured in an earlier column) the "Computational Complexity Blog."

In his guest column, Bill answers our questions on his experience writing for a theory blog with a very large and engaged community of readers, he tells us about his sources of inspiration, and he highlights two posts from his archives: one on open problems in mathematics and another on using SAT solvers to get concrete bounds on extremal combinatorics problems.

COMPUTATIONAL COMPLEXITY

A Conversation with Bill Gasarch

(If you are reading this in pdf then you can click on the links in this article by clicking on the "*here*" of "*see here*".)

Q: Bill, thanks for taking the time for this conversation. Can you tell us how your collaboration with Lance Fortnow on the Computational Complexity Blog got started?

Lance started complexityblog on August 22, 2002. Lance invited me to do a Complexitycast with him (see here). That went well. He went on vacation in January of 2006 and asked me to guest post for the week. My first guest post was titled *Are you a Luddite* (see here and follow up posts see here and see here). That post went well!

I began doing a few more guest posts. On March 25, 2007 Lance suddenly decided that he said all he wanted to say, so he retired (temporarily as it turned out) from blogging (see here).

He got several blog posts and emails saying that the blog SHOULD go on. I was the only person in the intersection of WANT TO DO IT and COULD DO IT. My first post as a non-guest blogger was on March 30, 2007 and just said *I am the new Complexity Blogger* (see here).

My first real post was on The Continuum Hypothesis since Paul Cohen had died recently (see here). I revisited this topic in 2020 (see here).

Q: Your writing style is very idiosyncratic, and when I read a post in the Computational Complexity Blog I can always tell if it was written by you or by Lance. Do you have any inspirations or models for your writing?

I list some of my influencers.

1. My hobby is comedy and novelty songs (I have a large collection) so I have absorbed some from that realm. Hence I keep it light (with pointers to more serious material), try to use clever wordplay, and know when to stop (its a maxim among comedians that you should tell the same joke twice, but no more than that).

2. Lance obviously influences me. He started the blog and set the tone for it.

3. The following authors all influenced me

   (a) In High School I read many books by Martin Gardner on recreational math (I find the difference between recreational math and serious math to be either thin or non-existent). I liked his style and content. I re-read some of his book for my book review column as an adult.

   (b) As an adult I read the Brian Hayes's *Group Theory in the Bedroom* which is not as sexy as it sounds, but is excellent at explaining math to the layperson.

   (c) Ian Stewart's book *Ian Stewart's Cabinet of Mathematical Curiosities* stands out for style since some chapters are long, some are short, some are funny, some are serious, some are doing math, some are commenting on math and some are not-quite-math. This influences me to NOT feel the need to be uniform.

   As an example of an entry that was short, funny, and not-quite-math, here is a poem from the book:

   A challenge for many long ages
   Had baffled the savants and sages
   Yet at last came the light
   Seems that Fermat was right
   To the margin add 200 pages

   (While trying without success to find the origin of that poem I found many great poems about Fermat's last theorem. See here.)

   (d) I read Doug Hofstadter's *Godel-Escher-Bach* in the summer between undergrad school and graduate school. I knew JUST ENOUGH logic to understand it but NOT SO MUCH as to be bored. I try to hit that sweet spot in my blogs as well.

   (e) This is not an author but a pointer: I wrote book reviews of books by some of the above authors.

      i. I wrote joint book review of books by Gardner, Hayes, and Stewart. See here.

      ii. I wrote a book review of *Martin Gardner in the Twenty First Century* which is about serious math he inspired. See here.

      iii. Note to Self: Reread *Godel Escher Bach* and write a review of it.

4. My Darling knows enough computer science and math to know what I am talking about (she has a Masters Degree in Computer Science) but not

enough to have drunk the Kool-aid. This has influenced some posts. The most obvious is that when I told her the Banach-Tarski Paradox she declared *Math is broken!* She may be right. The BT paradox was the subject of a blog post (see here) and has been mentioned in other posts.

Q: You often solicit reader's opinions or feedback on your posts, and you engage with them. Can you recall some memorable exchange that took place following one of your posts? Has your research ever been influenced by such discussions?

The comments I get inspire me to READ up on some topics, which helps my research indirectly. More likely what I get out of the comments is

1. material for my open problems column (I am currently the SIGACT News Open Problems Column Editor),
2. surveys,
3. books to read and write review of,
4. problems for the Maryland High School Math Competition,
5. blog posts which will give me
   (a) material for my open problems column (I am currently the SIGACT News Open Problems Column Editor),
   (b) surveys,
   (c) books to read and write review of,
   (d) problems for the Maryland High School Math Competition,
   (e) blog posts which will give me ...

Even though I was trained as a mathematician I will go against that training and give some *examples*. I give summaries of blog posts in italics with a pointer to the original post, and then some comments on the comments.

> *(See here) Alice, Bob, Carol each have an n-bit number on their foreheads and they want to know if the sum is* $2^n - 1$. *They can do this with n bits of communication. Can they do the problem with less bits of communication? This sounds like it could be a FUN problem to tell your undergraduates about. Chandra-Furst-Lipton [1] showed that, for large n, they can do it in* $\sqrt{n}$ *bits. The proof uses large 3-free sets (from Ramsey Theory) which I find fun, but your typical undergraduate might not. Is there a way to make this problem FUN by finding a way to do it with (say)* $\frac{n}{10}$ *bits but in a way undergraduates can understand?*

One of the comments gave an elementary $\frac{n}{2} + O(1)$ solution. Now that I have a starting point I will write an open problems column where I ask how well we can do using elementary methods. Lower bounds are impossible here since *elementary methods* is not rigorous; however, upper bounds would be great. I also (easily) extended the solution to k people and $\frac{n}{k-1} + O(1)$.

> 2) *(See here) Hilbert's 10th problem is to (in todays terms) find an algorithm that will, given a poly $p(x_1, \ldots, x_n) \in \mathsf{Z}[x]$, determine if there is a solution in Z. From the work of Davis, Putnam, Robinson, and Matiyasevich the problem is known to be undecidable. Let $H10(d, n)$ be the problem where the polynomial is of degree d and has n variables. There should be a grid of (d, n) saying, for each entry, if its undecidable (U), decidable (D), or unknown (UK). But there is not. Darn.*

The comments on this blog inspired me to do a survey of what is known, which appeared in the BEATCS algorithms column [2]. A later version is on arxiv (see here). An email from a reader was very enlightening and I quoted it in the arxiv version:

> *Timothy Chow offered this speculation in an email to me: One reason there isn't already a website of the type you envision is that from a number-theoretic (or decidability) point of view, parameterization by degree and number of variables is not as natural as it might seem at first glance. The most fruitful lines of research have been geometric, and so geometric concepts such as smoothness, dimension, and genus are more natural than, say, degree. A nice survey by a number theorist is the book Rational Points on Varieties by Bjorn Poonen [4] Much of it is highly technical; however, reading the preface is very enlightening. Roughly speaking, the current state of the art is that there is really only one known way to prove that a system of Diophantine equations has no rational solution.*

Q: I know you are quite interested in mathematics education and about getting K-12 kids interested in mathematical thinking and research. What role do you think that blogs can play in getting young people interested in mathematics?

This has two answers.
1) If a K-12 student reads my blog (probably High School, though I do know one 9 year old who reads it) then since it's light and we highlight the ideas it may

inspire them to read something more serious, or to contact Lance or I (some have contacted me).
2) Because I write the blog, I come up with ideas for High School projects. Because I am good at coming up with ideas for High School projects, I write the blog. I end up with many projects for High School Students.

Q: Can you highlight one post from the past and tell us about it?

I will highlight a few posts.

1) My advisor Harry Lewis emailed me that his 9 year old granddaughter Alexandra wants to know what happens when you get a Millenium prize so that she will be ready.

This inspired some silly and serious thoughts:

1. How likely is it that Alexandra will resolve P vs NP (or if she is rebellious, the Navier-Stokes equation)? Alas, not likely.

2. How much progress as a community have we made on P vs NP? Not much. That topic has been blogged about and discussed a lot before, so no need to rehash that topic.

3. Erdős has said of the Collatz Conjecture that
   Mathematics is REALLY not ready for this problem

   Alexandra and Erdős jointly inspire the following:

The post (see here) was a historical tour-de-force of open or previously open problems in mathematics. The most intriguing was the three problems of antiquity: can you, with just a ruler and a compass (as a kid I wondered why knowing what direction north was would help with geometry) trisect an angle, double a cube, or square the circle. When it was posted

Mathematics was not ready for this problem.

I then went through other math problems with the question *was math ready for it when it was posed?* The problems were: FLT, Completeness of Peano Arithmetic, the Continuum Hypothesis, Hilbert's 10th problem, The Four Color problem, Poincare's Conjecture, The Erdős Distance problem, The Collatz Conjecture (spoiler alert: Math is still not ready for it), Ramsey of 5 (not ready but sadly, not that interesting), and the Twin Primes Conjecture.

The post ended with the note that Alexandra was going to work on Collatz over the summer, and I wished her luck.

The post inspired me to read the book *Tales of the Impossible: The 2000 year quest to solve the mathematical problems of antiquity* by David Richeson, which was very enlightening. For my review of it in SIGACT News see here.

2) An $n \times m$ grid is *c-colorable* if there is a map from $[n] \times [m] \to [c]$ so that there are no rectangles where all four corners are the same color. I was working on the following problem (with co-authors Stephen Fenner, Charles Glover, Semmy Purewal): for which n, m, c is $n \times m$ c-colorable? We had determined exactly which grids were 2-colorable. We had determined exactly which grids were 3-colorable. We had reason to think that 17×17 IS 4-colorable. But we could not prove it. On November 30, 2009 I posted about the problem on the blog and:

offered a bounty of $ 289 to the first person to email me a 4-coloring of the 17×17 *grid. (See here.)*

Brian Hayes saw this and popularized the challenge in his column. His column has far more readers than mine does, so this got the problem more out there. Several people told me *just throw a SAT SOLVER at it*. Those who tried had no success. Until...

In 2012 Steinbach and Posthoff [5, 6, 7] obtained the coloring (and a few others that I needed) and I happily paid them the $289.00. I blogged about it (see here). I was thus able to solve exactly which grids were 4-colorable. The problem of 5-colorability seems to be beyond todays technology and might always be.

This is the most direct case of me posing a problem on the blog and getting it answered. For the final paper see here.

3) On April 1, 2010 I posted a manifesto that the theory community should STOP proving that our techniques won't suffice to solve certain problems (e.g., oracles, natural proofs) and PROVE SOMETHING. I laid down some problems where progress seemed possible.

1. Prove that NP is different from time $2^{O(n)}$.
2. Determine how NL and P compare.
3. Determine how deterministic primitive recursive and nondeterministic primitive recursive compare.
4. Determine how deterministic finite automata and nondeterministic finite automata compare.

I stated that these problems should be solvable with the methods available to us since neither oracles nor natural proofs rule out current techniques. So progress is possible.

The astute reader of this column may notice two things: (a) all four open problems mentioned are not open at all but it is well known what the answers are, and (b) this was posted on April fools day. One may also note that YES, they can be solved with current methods.

I am particularly proud of this post since it has become a cliche to post on April fools day that some *open* problems have been *closed*. I turned that around: I posted that some *closed* problems are *open*.

Q: Is there something else that you would like to share with our readers?

Some random thoughts:

1. Early on Lance told me to NOT worry about comments. Engage YES, but do not write a post thinking *this will get a lot of comments!* Unlike todays journalists (or perhaps social media aggregator) we are not paid by-the-click. We write what we find interesting and hope others will, but are not obsessed with that part. To quote Ricky Nelson's song Garden Party *you can't please everyone, so you've got to please yourself.* For the video on you tube of that song, see here.

2. Blogs are a wonderful place to exchange ideas without referees or program committees getting in the way.

3. Clyde Kruskal and I have written a book based on some of my blog posts. We took those that made a point about Math or CS, and then did some Math or cs to illustrate the point. The essays in the book are polished and completed versions of the posts, or in some cases a set of posts. *Problems with a point: Exploring Math and Computer Science*. See here.

4. While the blog is called *complexityblog* we do not feel constrained by that. Many topics are in our scope. I am a fan of Ramsey Theory, so that comes up a lot. Non-theory topics in computer science, issues about academic computer science or academia in general are certainly fine topics. We sometimes comment on current event. When Jimmy Buffett passed away I had a blog (see here) about songs that have a *contradiction* (thats is relevant for logic!) between what the *lyrics say* and what people *think they say* since Jimmy Buffett's *Margaritiville* is a great example.

5. This column has 25 links in it. Is that a BEATCS record? The blog post with the most links was titled *Disproving that Myth that many early logicians were a few axioms short of a complete set* (see here) which had 71 links.

References

[1] A. Chandra, M. Furst, and R. Lipton. Multiparty protocols. In *Proceedings of the Fifteenth Annual ACM Symposium on the Theory of Computing,* Boston MA, pages 94–99, 1983. http://portal.acm.org/citation.cfm?id=808737.

[2] W. Gasarch. Hilbert's tenth problem for fixed d and n. *Bulletin of the European association of theoretical computer science (BEATCS)*, 133, 2021. See the arxiv article LINK.

[3] W. Gasarch and C. Kruskal. *Problems with a point: exploring math and computer science*. World Scientific, 2019.

[4] B. Poonen. *Rational points on varieties*, volume 186 of *Graduate studies in mathematics*. American Mathematical Society, 2017.

[5] B. Steinbach and C. Posthoff. Extremely complex 4-colored rectangle-free grids: Solution of an open multiple-valued problem. In *Proceedings of the Forty-Second IEEE International Symposia on Multiple-Valued Logic*, 2012.

[6] B. Steinbach and C. Posthoff. The solution of ultra large grid problems. In *21st International Workshop on Post-Binary USLI Systems*, 2012.

[7] B. Steinbach and C. Posthoff. Utilization of permutation classes for solving extremely complex 4-colorable rectangle-free grids. In *Proceedings of the IEEE 2012 international conference on systems and informatics*, 2012.

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 141

Reusing Space: Techniques and Open Problems

Ian Mertz[‡]

Abstract

In the world of space-bounded complexity, there is a strain of results showing that space can, somewhat paradoxically, be used for multiple purposes at once. Touchstone results include Barrington's Theorem and the recent line of work on catalytic computing. We refer to such techniques, in contrast to the usual notion of reclaiming space, as *reusing space*.

In this survey we will dip our toes into the world of reusing space. We do so in part by studying techniques, viewed through the lens of a few highlight results, but our main focus will be the wide variety of open problems in the field.

In addition to the broader and more challenging questions, we aim to provide a number of questions that are fairly simple to state, have clear practical and theoretical implications, and, most importantly, that a newcomer with little background experience can still sit down and play with for a while.

The first subroutine I coded, learning the principles of object oriented programming in a CS 101 course, was the swap function. Every coder knows it by heart: three lines—their syntax is nearly universal across all languages—and only using the two input variables in question plus one additional:

```
temp = x
x = y
y = temp
```

Years later I was adding the same function as part of the preamble to a larger assignment, when the TA gave me a puzzle: can you swap two bits without the temp register? Yes, and with no increase in the length of the program:

[*]Centre for Discrete Mathematics and its Applications (DIMAP), University of Warwick, UK. Email: ian.mertz@warwick.ac.uk.

[†]The author received support from the Royal Society University Research Fellowship URF\R1\191059 and from the Centre for Discrete Mathematics and its Applications (DIMAP) at the University of Warwick.

```
x = x ⊕ y
y = x ⊕ y
x = x ⊕ y
```

While not as ubiquitous as the standard program, no shortage of experienced coders have encountered this problem before; indeed it is one of many gateways to the wide world of bit-tricks. But like all good hammers, the answer, an elegant improvement to one of the standard subroutines in all of computer science, begs one to go looking for nails, or, better still, more hammers of a similar ilk.

1 TCS Wants YOU (To Reuse Space)

1.1 Who, me?

Why should we think about reusing space? Consider some clickbait-ized headlines for the successes of the field thus far:

- any function can essentially be computed using just a three-bit memory
- access to a hard drive can be more powerful than non-determinism, even when it's full
- our central approach to separating P from L contains a fatal flaw
- the max flow rate is not an upper bound on the size of messages that can be sent through a network

These are sensational highlights, and for those in space-bounded complexity or similar fields they may warrant scrutiny on their own merit. But in this work, rather than giving a status report on a distant field, I want to invite a broader audience to consider coming into the fold themselves, and so our focus will be slightly different.

The survey will be split into two parts. For the readers in search of hammers, we will give an overview of the major techniques in the field, which should be sufficiently general and straightforward so as to spark the reader's imagination. And for the readers who enjoy a good nail, we will present a broad swath of open problems in the field, from puzzles to be worked on over an idle lunch to the titanic central problems in space complexity.

Part I will give some background into the existing techniques on reusing space. This part will follow more of the typical pattern of a survey, but with all proofs kept at a fairly high level with minimal definitions or details; we eschew all formal preliminaries of basic objects—e.g. circuits, branching programs, etc.—in favor

of brief descriptions of relevant characteristics, as our goal is simply to give the reader a taste for past arguments. We also include a number of exercises to the reader[1], whose answers can be found in the appendix at the end.

In Part II we lay out a number of open problems in the world of reusing space. These sections will give a flavor of the problem itself, known results and techniques, possible hindrances, and consequences of solving them. Again the details will be left fairly light, focusing on breadth rather than depth in order to appeal to many types of problem solvers, as well as to avoid personal biases as much as possible, although in this I admit to have largely failed.

1.2 Reducing via Reusing (beyond Recycling)

Per the title, we focus on *reusing*, rather than simply *saving*, space, even though proving upper bounds through space-saving algorithms is our ultimate aim. Let us disambiguate these terms now.

What could reuse mean, beyond reclaiming space as it becomes available? The latter is something we understand quite well. When we study the space complexity of some function, we naturally focus on what the algorithm *needs* to remember during the computation, and so we design algorithms with the aim of remembering, at any point in time, as little information as possible.

In this survey we study a slightly different question: can we store *lots* of information but do so using *much less* space? By this we do not mean a simple question of compression, for which the same information theoretic bottlenecks still hold sway. We ask a more suggestive question: *can we use space for two things at the same time?*

We prime the reader to this possibility by mentioning two distinct uses of space as a resource writ large: *storage* and *work*. At any moment in a computation, there may be information concerning the global computation that must be written down for future use, while simultaneously there are local computations that must be performed with the aid of the tape as well.

The question of whether these need occupy separate places in memory—what one could call the *composition* question for space—may at first glance appear trivially true, but this is not the case. In Section 2 we will see how memory can be used for two such purposes at once by analyzing the proof of Barrington's seminal result [Bar89], as adapted from a follow-up work of Ben-Or and Cleve [BC92].

[1] Typically we write "this is left as an exercise to the reader" to denote either something trivial or something technical but lacking in key ideas; in any event, something to be glossed over. In this survey it means precisely the opposite: readers who want to get comfortable with the techniques presented are highly encouraged to try them out and check their work.

1.3 Our test module: catalytic computing

Our goal in this survey is to appeal to a broad audience within theoretical (and possibly even applied) computer science about the intriguing mysteries of reusing space. Thus as much as possible we will refrain from narrowing the focus to one model or another.

However, we cannot avoid introducing a model of space which has been intertwined with such questions for many years now, and which seems a natural first stop when studying any questions about reusing space: *catalytic computing*.

Consider the two uses alluded to in the previous section: storage space and work space. How can we focus on separating out these uses? The simplest way is to imagine a situation where the work space we seek to use stores information that is completely unrelated to any computation at hand; in fact, we go a step further and consider a work tape populated with *arbitrary* bits, and see whether or not such a tape can still be useful for computation.

To focus on this question, imagine a space-bounded machine in the usual sense, but now we also give it access to a second work space, which we call the *catalytic tape*. While our main work tape is initialized to be empty, our catalytic tape is initialized to be full; what information fills the tape is arbitrary and out of our control, and while we allow the machine to use the catalytic tape however it chooses, we stipulate that the machine should return this memory to its original state at the end of the computation.[2]

Seeing if clever use of the catalytic memory allows us to exceed the power of a typical bounded space machine is one clear way to test our ability to reuse space without simply reclaiming memory used for past computations. In Section 3 we will mention some key results in catalytic computing, and more importantly the techniques used in these results, which shows that such reuse is not only possible but quite powerful.

1.4 Many flavors of questions

Given these preliminary motivations, there are many types of problems we can ask. We loosely group these into four categories. In Section 4, we tackle the basic premise of our field by looking to apply the techniques of reusing space to answer questions in space-bounded complexity. In Section 5, we ask how the catalytic computing model compares with traditional complexity classes, both space-

[2]The term catalytic refers to a catalyst in chemistry, which is a chemical unrelated to, and ultimately preserved in quantity by, a given reaction, but whose presence is nevertheless necessary for the reaction to occur. We also note that there are multiple unrelated definitions of "catalytic computing" circulating in the CS literature, including one for network systems and another in quantum computing; all of these models, of course, are named for the same physical phenomenon.

bounded and otherwise. Section 6 asks similar questions but comparing catalytic classes to one another in an attempt to flush out a parallel structural theory of space. Finally in Section 7 we go beyond space-bounded complexity classes and consider when the techniques we have seen in this work may apply to alternative computational models; in this section in particular I invite the reader to think about their own research and build novel connections to the framework of reusing space.

1.5 The purpose of this survey

Writing a survey article is a chance for the author to bring a new and (personally) exciting field to the reader's attention, and to isolate the central and furthest reaching successes therein. It can appeal to the utility of such results and proofs, ones the reader, a researcher with a full schedule and their own field and goals, may have never even heard of. It is, then, a way to plant the seed of interest while respecting the reader's time.

Why, then, do we focus not on the use to the reader, but rather to beg them to drop their own work and spend time on an alien set of questions? One answer is that the former duty has already been discharged in the 2016 edition of this column by Michal Koucký [Kou16]; while there have been exciting results since then, another review only seven years later is hardly warranted. We will spend Part I of our survey covering some of the basics that appeared in this excellent work, but it will be for the sake of definition and intuition; for all other purposes, I refer you to therein.

Another answer, which was alluded to in the preamble, is that the nascent field finds itself in a very fortunate position: many of the existing open questions in reusing space can be described, motivated, and attacked with very little background. Some revolve around basic arithmetic; some involve drawing small graphs. There are no shortage of questions that ask for a slight twist on some fundamental theorem from an undergraduate complexity course. Hence they may be appropriate for those looking for "toy" (but still consequential) problems to play with, such as early career researchers—I have even given some problems to undergraduates in the past—or those who enjoy doing puzzles at dinner.

Lastly, I hope that beyond the specific problems at hand, the reader will take with them, back to their own specific subfield, the question of how one might use space, or indeed any other resource, in more than one way at once, and to what end. The faith that motivates this survey is that the question of reusing space will both benefit and benefit from researchers from a wide variety of fields; I will present one approach to solving one type of question, but more than pushing this particular angle forward, what the field needs, and potentially offers, is a greater variety of approaches to, and understandings of, its central tenets.

PART I: WHAT WE KNOW (THE BASICS)

2 An introductory example: Barrington's Theorem

2.1 Statement and proof

In order to calibrate ourselves to the task of reusing space, let us see one simple but foundational example of what this actually looks like.

Our adversary will be the circuit, a classical model of computation wherein an input x is fed bit by bit into a network of AND, OR, and NOT gates. Let C be a circuit taking n inputs; to simplify matters, we remove every OR gate from C using de Morgan's laws, and assume every AND gate takes two inputs.

"Theorem": C can be computed using effectively three bits of memory.

The statement "C can be computed using three bits of memory" is, of course, assuredly false; the proof of our "theorem" will fall short of this, due to a variety of technical considerations, including uniformity, counters for runtime, etc. However, it is correct in a moral sense, one which can be converted into useful statements and algorithms, and, more important at the present, provides the basis for our view of reusing space throughout the rest of this survey.

The proof is self-contained and only relies on basic modular arithmetic, and thus we state the proof first and save the background and intuition for the rest of the section.

Proof. Our argument will be by induction on the gates of the circuit C. Let (R_0, R_1, R_2) be our three bit memory, initialized to $(0, 0, 0)$, and fix the input x under consideration. Our inductive statement is as follows:

Lemma 1. *Let (R_0, R_1, R_2) be in some state $(\tau_0, \tau_1, \tau_2) \in \{0,1\}^3$, and let g be a gate in C which takes value v_g on input x. Then for $i \in \{0, 1, 2\}$, there is a program $P_g(i)$ which transforms the memory as follows:*

$$\begin{aligned} R_j &= \tau_j \oplus v_g \quad j = i \\ R_j &= \tau_j \qquad\quad j \neq i \end{aligned}$$

We see that this is sufficient to compute f. Our first recursive call will be to the output gate *out* of C, say $P_{out}(0)$, and here (τ_0, τ_1, τ_2) is the initial blank tape, i.e. $(0, 0, 0)$; thus at the end of the computation, R_0 will be in state $0 \oplus v_{out} = f(x)$.

So now we take up this task. From here out out we view everything through the lens of arithmetic modulo 2, meaning + denotes $\oplus$. We use notation $R \mathrel{+}= v$ to mean $R \leftarrow R + v$, and so our goal is to design, for every gate $g \in C$ and

$i \in \{0, 1, 2\}$, a program $P_g(i)$ which computes $R_i \mathrel{+}= v_g$ while leaving all other memory untouched.

The base case is simple enough: if g is one of the inputs to the global function f, say x_j, then we can simply add the relevant input to whichever register we please and we are done:

$$R_i \mathrel{+}= x_j$$

Now let g be an internal gate in the circuit, i.e. either NOT or AND. NOT is simple enough: let $g = \neg h$, let v_h be the value of h, and by induction let $P_h(i)$ be a program computes $R_i \mathrel{+}= v_h$. Then simply running $P_h(i)$ and then adding 1 yields

$$R_i = \tau_i + v_h + 1 = \tau_i + \neg v_h = \tau_i + v_g$$

Thus our last case is to compute $g = g_1 \wedge g_2$, and without loss of generality we focus on $P_g(0)$, as $P_g(1)$ and $P_g(2)$ can be accomplished by relabeling.

Let g_1 and g_2 take values v_1 and v_2, and by induction let $P_1(1)$ and $P_2(2)$ be programs which send R_1 to $\tau_1 + v_1$ and R_2 to $\tau_i + v_2$, respectively. The following program sends R_0 to $\tau_0 + v_1 v_2$; the relevant memory states are listed inline, with brackets separating out the previous memory state and the new additions:

1.	$P_1(1)$	R_1	$= [\tau_1] + [v_1]$
2.	$R_0 \mathrel{+}= R_1 R_2$	R_0	$= [\tau_0] + [(\tau_1 + v_1)(\tau_2)]$
			$= \tau_0 + \tau_1\tau_2 + v_1\tau_2$
3.	$P_2(2)$	R_2	$= [\tau_2] + [v_2]$
4.	$R_0 \mathrel{+}= R_1 R_2$	R_0	$= [\tau_0 + \tau_1\tau_2 + v_1\tau_2] + [(\tau_1 + v_1)(\tau_2 + v_2)]$
			$= \tau_0 + \tau_1 v_2 + v_1 v_2$
5.	$P_1(1)$	R_1	$= [\tau_1 + v_1] + [v_1]$
			$= \tau_1 \quad \checkmark$
6.	$R_0 \mathrel{+}= R_1 R_2$	R_0	$= [\tau_0 + \tau_1 v_2 + v_1 v_2] + [(\tau_1)(\tau_2 + v_2)]$
			$= \tau_0 + \tau_1\tau_2 + v_1 v_2$
7.	$P_2(2)$	R_2	$= [\tau_2 + v_2] + [v_2]$
			$= \tau_2 \quad \checkmark$
8.	$R_0 \mathrel{+}= R_1 R_2$	R_0	$= [\tau_0 + \tau_1\tau_2 + v_1 v_2] + [(\tau_1)(\tau_2)]$
			$= \tau_0 + v_1 v_2 \quad \checkmark$

which completes the proof, as $R_0 = \tau_0 + v_g$ and $R_i = \tau_i$ for $i = 1, 2$. □

The statement and proof above are adaptations of the seminal works of Barrington [Bar89] and Ben-Or and Cleve [BC92]. We give a more exact statement at the end of this section, turning now to focus on a higher level understanding of their technique.

2.2 Back to basics

Let us now rewind and build back towards our "theorem" from the ground up. Our goal is to demystify the proof we have just seen, and in the process to begin thinking about how these principles can be be extended.

2.2.1 Circuits and space

In the world of Boolean functions, circuits are one of the most fundamental computation models, and are universal in the sense that any function has a corresponding circuit. However, we typically consider the class of functions f which are *efficiently* computable by circuits, both with respect to *total time*, measured by the number of gates, and *parallel time*, measured by the longest path from any input wire to the output of the circuit. We say the *size* of a circuit is the number of gates, while the *depth* of a circuit is the length of the longest input-output path.

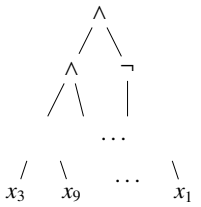


Fig.: circuit

More specifically, consider a family of functions $f = \{f_n\}_{n\in\mathbb{N}}$, each f_n taking in n input bits. Let f be computable by a family of circuits $C = \{C_n\}_{n\in\mathbb{N}}$, where each C_n has size s_n and depth d_n. Our goal will be to design a machine which computes f_n using the minimal possible space.

As should be clear from the initial statement, we will not be rigorous about which operations are allowed and such; we will assume that our machine has knowledge of C_n, and that at each step it makes use of all available information—by which we mean the circuit, the input, and the current state of our memory—to progress in the computation.

We begin with a sanity check: each f_n can be computed without reusing any space, and the space complexity corresponds to the size of C_n.

Claim 1. *f_n can be computed in space s_n.*

Proof. We assign to each gate in C_n a spot on our work tape, and progress through the circuit in order writing down the value of each gate. If the gate uses any input bits directly we take them from the input tape, while if it takes previously computed gates as input we read their values from the work tape. The final gate will compute the output of C_n, and thus the value of f_n. □

This procedure is simplicity itself, and yet clearly wasteful. As soon as some internal gate of C_n becomes irrelevant to future computation, we could reclaim this space to use for later gates.

Alternatively, we can throw away information about a gate even if it is required for later use, as long as we are willing to pay to recompute it in the future, perhaps when more space is at our disposal.

Can this logic be exploited when we know nothing about C_n besides its size and depth? Indeed we can, if we consider how and when computations are reused. Thus we can reduce the space complexity from the total runtime of C_n to its parallel runtime, from size to depth.

Claim 2. *f_n can be computed in space d_n.*

Proof. For each gate $g \in C_n$, we define the *level* of g to be the length of the longest finite path any input of the circuit takes to get to g. Since every input takes at most d_n steps to reach the output of the circuit, every gate is at level at most d_n, and the output gate is at level d_n exactly.

We will inductively prove that any gate at level k can be computed with space k, thus proving the claim by considering the output gate at level d_n. For any gate g at level 1, the inputs to g simply come from the input itself, and thus we can write down the value of g directly from considering the input tape.

Now for gate g at level k, let g_1 and g_2 be the inputs to g, each of which occur at some level strictly less than k. To compute g we first compute g_1, which by induction can be done in space at most $k - 1$. Now we erase the entire work tape save for the output of g_1, and we then compute g_2 in the free space on our tape. This again can be done in space at most $k - 1$, and combining this with the output of g_1 we saved earlier gives us space k to compute g itself. □

2.2.2 Constant space: who could expect it?

Here we see that a careful view of how the internal computation of a circuit works can point us towards lowered space. Can this be pushed further?

While it may seem bold to press on, we may take heart from observing that at any moment in time, only *two* bits of the work tape are relevant to the computation at hand, no matter where in the recursion we may find ourselves. Thus the true test of our resolve is to ask: can we compute f using only a *constant* amount of memory?

On the face of things, this task is manifestly impossible. It is certainly true that only two bits are needed at any moment in time; yet this is making a similar oversimplification as saying only the output gate of C_n is truly important for computing the value of f_n. Each internal computation is unimportant alone, but is vital for the next step, and as such it is not just which bits are relevant in *this* step, but also which bits will be relevant in *later* steps. For a concrete example, during the computation of g_2 in the proof of Claim 2, it may be correct to say the value of g_1

is irrelevant now, but to erase it would clearly be shooting ourselves in the foot in a moment's time.

Thus to begin, we must ask whether we can broaden our horizons in defining what it means to reuse space. So far we have limited ourselves to *reclaiming* space and using it as if it were fresh. A more daring idea is to reuse space *in situ*, or in other words to use space for more than one purpose *at the same time*.

Here we focus on two distinct uses of space during the computation: space as providing *inputs* to the current computation and space as *storage* for future computation. If a bit of memory could play both roles simultaneously—to again use our previous proof as a concrete reference, if we imagine that the space *storing* g_1 could be used in parallel for *computing* g_2—then our previous objection becomes weaker.

2.3 Proof redux

We are now ready to challenge our "theorem" once again. By building our ideas of how to reuse space from the ground up, we will once again reach the proof given at the beginning of this section.

Naturally we will once again attempt an inductive approach, computing the circuit gate by gate. The proof lies in two insights: first, in the way that it arrives at an *algebraic* definition for reusing space which can be turned into a suitable recursive statement; and second, in the way that definition's algebraic nature suggests the method by which the statement can be achieved, if one tries at every step to do what immediately brings them closer to the goal.

2.3.1 Idea 1: modular arithmetic

As suggested by the discussion from the previous section, we need to choose a recursive statement whereby g is successfully added in memory while not erasing, and in fact in a formal sense preserving, its current state.

The term "adding" in the previous statement should immediately call to mind one such potential definition: *XORing* the value of g to memory. To figure out our exact statement, let us put ourselves in a moment within the computation.

With respect to memory as storage, we will imagine our three-bit memory is in some state $(\tau_0, \tau_1, \tau_2) \in \{0,1\}^3$, the exact value of which will be dictated by the recursion thus far; rather than subject ourselves to working out an exact statement for the values τ_i, we consider them to be arbitrary and out of our control.

With respect to memory as computation, our goal will be to send one of these bits, say τ_0, to the value $\tau_0 + v_g \mod 2$, where again v_g is the value of g in question. This statement suggests that recursively we should assume we have the ability to

do the same below us, namely that we can send τ_i to $\tau_i + v_h \mod 2$ for any gate h which is an input to g.

Furthermore, in order to respect the stored values τ_i, we will design our program in such a way that after computing v_g into R_0, the values stored in R_1 and R_2, namely τ_1 and τ_2, are left unchanged, and whereby recomputing v_g allows us to recover τ_0, i.e. to restore our last piece of memory R_0.

This brings us squarely to our choice of recursion statement as given by Lemma 1. Given this concrete goal, and with the assurance that proving it is sufficient to proving the "theorem", we may be encouraged by how simply the base case, i.e. the input layer, as well as the case of $g = \neg h$, can be dismissed with.

This only leaves us with proving the recursive statement for $g = g_1 \wedge g_2$. The reader is invited to pause here and take stock by attempting to do so themselves; it is the certainly the only tricky part of the argument once Lemma 1 is formulated, but it can be accomplished with a little trial and error.

2.3.2 Idea 2: handling multiplication

It seems that the only place to start is adding $g_1 \wedge g_2$ to τ_0 in any way we can, and then seek to fix things up from there. The most natural way to do this is to execute $P_1(1)$ and $P_2(2)$, and add the AND of the resulting memory to τ_0. Viewing AND as multiplication modulo 2 and expanding the product, we get

$$\tau_0 + (\tau_1 + g_1)(\tau_2 + g_2) = \tau_0 + \tau_1\tau_2 + \tau_1 g_2 + g_1\tau_2 + g_1 g_2$$

which contains exactly the terms we want, namely $\tau_0 + g_1 g_2$, as well as three junk terms, namely $\tau_1\tau_2 + \tau_1 g_2 + g_1\tau_2$.

To remove these terms, let us start with $\tau_1 g_2$, which suggestively does not contain g_1. Executing $P_1(1)$ sends $\tau_1 + g_1$ to $\tau_1 + g_1 + g_1 = \tau_1$, and thus the AND of our two work bits is $\tau_1(\tau_2 + g_2) = \tau_1\tau_2 + \tau_1 g_2$. We add this to our target memory.

We can take care of $g_1\tau_2$ similarly, executing $P_2(2)$ to remove g_2 and $P_1(1)$ to get back g_1, and then adding $R_1R_2 = (\tau_1 + g_1)\tau_2$ to our target register. Thus our memory contains

$$(\tau_0 + g_1 g_2 + \tau_1\tau_2 + \tau_1 g_2 + g_1\tau_2) + (\tau_1\tau_2 + \tau_1 g_2) + (\tau_1\tau_2 + g_1\tau_2) = \tau_0 + g_1 g_2 + \tau_1\tau_2$$

at which point we can simply reset our "external memory" by executing $P_1(1)$ one last time, and then adding the AND of R_1 and R_2 one last time seals the deal.[3]

After all is said and done we have not only added $g_1 \wedge g_2$ to τ_0, but in fact have set R_1 and R_2 back to their original values, thus fulfilling the other requirement of our recursive call.

[3] Note that our original proof was more efficient in terms of recursive calls; our only instructions are recursive calls and R_0 += R_1R_2, meaning the order in which we add our terms is irrelevant.

2.4 Afterword: next steps

The techniques involved in this section are crucial to understanding many of the arguments at the forefront of reusing space. Thus we will use this opportunity to give the reader their first exercise; since the solution can be found earlier in this section, the solutions manual will only contain an alternate analysis of the multiplication program.

Exercise 1. *Watch an episode of Gilligan's Island*[4]*, then come back and attempt to reprove our "theorem".*

We also dismiss with the tedious use of "theorem" by stating the actual result in question. This statement, the true form of Barrington's Theorem [Bar89], follows by a more optimized variant of Lemma 1, while Ben-Or and Cleve [BC92] devised Lemma 1 essentially as stated to prove a more general theorem.

Space-bounded computation has a very well-studied syntactic model to call its own: branching programs. A branching program is an directed acyclic graph with a start (source) node and two potential end (sink) nodes, one for each output of the function, where each node is labeled with an input variable x_j and has one outgoing edge for each potential value of x_j, i.e. 0 or 1. Computation is done in the natural way, starting at the source and at each node following the edge whose label agrees with the value of the x_j labeling the node itself, until we reach an output node whose value is declared the output of the function.

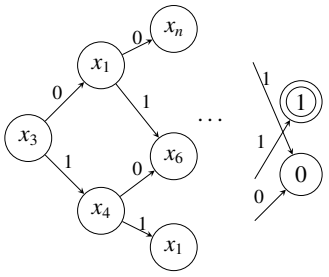


Fig.: branching program

While space is formally captured by the log of the size of the branching program, i.e. the number of bits needed to keep track of where in the graph we are at each moment in time, we have more fine-grained notions that are relevant. We say the program is *layered* if the nodes can be arranged into layers, starting with the source at layer 0, such that the edges coming out of any node at layer i go to nodes at layer $i + 1$. In this case we can speak of time and space as being the length and width of the program, i.e. the number of layers and the largest size of any layer, respectively.

[4]Any activity lasting a half hour or more and which does not involve mathematics will do; I suggest a cup of tea and a nice book. However, as this strategy was originally taught to me as "the Gilligan's Island method" (by a professor whose age was betrayed therein) I have preserved it as such.

Theorem 1 (Barrington's Theorem)**.** *Let C be a circuit of depth d. Then there exists a layered branching program of length 4^d and width 5 computing the same function as C.*

Lemma 1 immediately gives Theorem 1 with width $2^3 = 8$ instead of 5 (as well as some loss in the length). It can also be used to prove a more general arithmetic statement, which was the original motivation and result of [BC92]. This leads in to one of the key ideas in the upcoming section, and so we encourage readers to attempt this generalization for themselves.

Exercise 2. *Let C be an arithmetic circuit of depth d, meaning a circuits whose inputs are from a ring $\mathcal{R}$ instead of $\{0,1\}$ and whose gates are $+$ and $\times$ over $\mathcal{R}$. Extend Theorem 1 to show that there exists a layered branching program of length 4^d and width $|\mathcal{R}|^3$ computing the same polynomial as C.*

3 A brief primer on catalytic computing

The proof in the previous section gave us a taste of how one could use full memory, represented in Lemma 1 by the arbitrary values τ_i in R_i, in a non-trivial way. We now broaden the scope of such ideas to discuss our central, although certainly not exclusive, model for testing the reuse of space: catalytic computation.

3.1 The basic definitions

We start with the definition of Buhrman et al. [BCK+14], who first introduced the concept of catalytic computation.

Definition 1. *A catalytic Turing Machine with space $s := s(n)$ and catalytic space $c := c(n)$ is a Turing Machine M with two read-write work tapes, which we call the work tape and the catalytic tape, which have lengths s and c respectively.*

In addition to the usual restrictions on space-bounded Turing Machines, M obeys the following additional property: for any $\tau \in \{0,1\}^c$, if we initialize the catalytic tape to τ, then on any input, M contains τ on the catalytic tape when it halts.

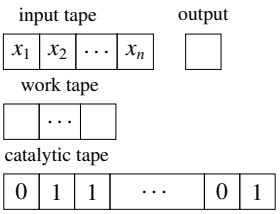


Fig.: catalytic Turing Machine

This definition gives rise to a natural complexity class, which is a variant of the ordinary class $\mathsf{SPACE}(s)$.

Definition 2. *The class* $\mathsf{CSPACE}(s, c)$ *is the set of all functions which can be computed by a catalytic Turing Machine which has space* s *and catalytic space* c*.*

$\mathsf{CSPACE}(s, c)$ sits between $\mathsf{SPACE}(s)$ and $\mathsf{SPACE}(s + c)$, with neither containment known to be strict in any interesting setting. Showing $\mathsf{CSPACE}(s, c)$ is strictly more powerful than $\mathsf{SPACE}(s)$ would be a validation of our ideas of reusing full space, even in the generic setting where we make no considerations of what the full memory actually contains.

The most well-studied variant of CSPACE is *catalytic logspace*, where s is logarithmic and c is polynomial. For most of this survey we frame our discussions around this class, but the reader should be aware that almost all results and problems we pose can be scaled up to pertain to other $\mathsf{CSPACE}(s, c)$ classes accordingly.

Definition 3. *The class* CL *is defined as* $\mathsf{CSPACE}(O(\log n), n^{O(1)})$.

Following our earlier discussion, CL sits somewhere between L and PSPACE. Current evidence, which we now turn to, suggests that both containments are strict.

3.2 Upper bounds

3.2.1 Compression: useful even when it fails

Let us disregard the techniques we saw in Section 2 and consider the definition of CSPACE at face value.

It is a natural knee-jerk reaction to think that $\mathsf{CSPACE}(s, c)$ must be approximately the same as $\mathsf{SPACE}(s)$ for any c. The only obvious approach to refuting such a statement would be applying some type of compression to the catalytic tape, and when one considers that such compression must succeed for any initialization of the catalytic tape—never mind the technical hurdles needed to implement such an approach—even this seems unlikely to help.

However, failure to compress such a string does not leave us with only free space s; it leaves us with free space s *plus* an incompressible string, hardly a trivial object to come by. Such a string may suggest many uses, but perhaps the first one that comes to mind is to use it as a source of entropy, i.e. randomness.

This insight can be pushed all the way through for simulating *randomized space*, due to a few peculiarities in how randomized space-bounded algorithms are defined. This proof [Lof] is unpublished as it was quickly subsumed by a very different argument, one which we turn to in the next subsection. We nevertheless reproduce it here because it is one of the few techniques which is known for catalytic space.

Theorem 2. BPL $\subseteq$ CL

Proof. Let us recall the definition of BPL: these are the functions f for which there exists a logspace machine B taking in an input x and which can generate a polynomial amount of randomness r, such that for every x, $B(x)$ outputs $f(x)$ with probability at least $2/3$ over all choices of r. Furthermore, we have the crucial restriction that B can only read each bit of randomness once; any bits of r that it wishes to use in the future must be stored on the logspace work tape.

Let B be a BPL machine, fix an input x to B, and without loss of generality assume B reads a bit of its random tape at every time step during its execution. To start thinking about derandomizing B, we must ask what a random string r needs to look like in order to be useful to B.

Nisan [Nis93] provided the following test for a collection of strings R. Run B on each $r \in R$ up to a fixed time step i, and partition up R based on which configuration σ—meaning the contents of the work tape, location of all tape heads, etc—B ends up in at this step. Essentially, Nisan's condition is that no matter which i and σ we look at, the $i+1$st bit should be fairly unbiased. This condition is checkable in logspace, and should it succeed then Nisan proves that a majority of strings in R will output the correct answer to $B(x)$.

We will let our catalytic tape be large enough that it can be broken into a collection R of candidate random strings. Using Nisan's criteria, we can use the normal work tape to check if the condition holds for each i and σ, and if it does then we simply run B on x using each $r \in R$ and take a majority vote. Note that in this case we never alter the catalytic tape, so we fulfill our additional requirement by default.

If this condition fails, then we can identify a timestep i and a configuration σ for which it fails. We let Γ be our set of strings from R which put B in configuration σ at step i, and by making $|R|$ sufficiently large we ensure that Γ has sufficiently many more strings with $b \in \{0,1\}$ than $\neg b$ in the $i+1$st location. Note that membership in Γ can be identified by running the BPL machine up to step i on our logspace worktape, and so by extension we can form a subtape T of the catalytic tape which exactly contains the $i+1$st locations of strings in Γ, with this tape containing, without loss of generality, polynomially more zeroes than ones.

Because of the severe imbalance of T, we can compress it in place and free up a polynomial number of cells.[5] At this point we can simulate our BPL machine B in a brute force manner, by using the empty cells of T to try every possible random string and take a majority vote. Afterwards, we save the answer on the smaller work tape, and before halting we run the inverse of our previous compression algorithm to return the catalytic tape to its original state. □

[5] We will not cover the details of this procedure. It is not difficult, but also not the style of argument we are concerned with in this survey.

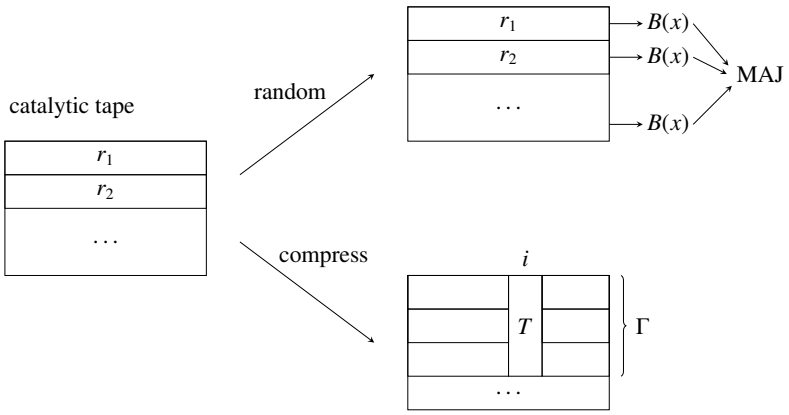


Fig.: compress-or-random argument

Throughout the rest of this survey, we refer to this proof as the *compress-or-random argument*. In thinking about the compress case, it seems like we have moved back from the goal of reusing space to that of simply saving space in the traditional sense. However, the real insight is coming from the random case, where the space that is being used for storage, i.e. the catalytic tape, is also being mobilized for a computational purpose.

3.2.2 Transparency and arithmetic

From the simple insight that incompressability gives us non-trivial power, we obtained a surprising use of catalytic machines. For this, we made essential use of the initial values stored in the catalytic tape. We now return to the ideas from Section 2 and take a different path to the power of full memory, one where we formally cancel out the contributions of the initialized catalytic tape without ever inspecting its values.

For this approach, we will define a toy model based on the statement in Lemma 1, in order to focus on the nature of our approach as both recursive and mathematical. For this section let $\mathcal{R}$ be a ring, i.e. a set of values and two operations + and $\times$ under which it is closed.

Definition 4. *A register program P with space $s := s(n)$ and time $t := t(n)$ is comprised of a set of s blocks of memory, or what we will call registers, $R_1 \dots R_s$, each of which can hold a single value from $\mathcal{R}$, plus a list of t instructions, each of which updates a single register by adding to its current value some polynomial*

over all the other registers.

$$R_i \leftarrow R_i + p(x_j, R_1 \dots R_{i-1}, R_{i+1} \dots R_s)$$

Often we will allow the program P to execute some other program P′ in place of an instruction. In this case, we will usually keep two separate notions of time: the number of basic instructions and the number of recursive calls.

Clearly this definition captures the algorithms defined in Lemma 1; for example, for gate $g = g_1 \wedge g_2$, our program $P := P_g(0)$ used three registers over $\mathbb{F}_2$, four basic instructions, and four total recursive calls to the programs $P_1(1)$ and $P_2(2)$.

These programs had an essential characteristic which made them useful to recursion as well as the ultimate task of reusing space: they worked even when all the registers R_i were initialized to ring values τ_i, and at the end of the computation they left all but one register untouched. This was given explicit attention by Buhrman et al. [BCK+14] for its use in building recursive procedures.

Definition 5. *A transparent register program P is one in which all registers R_i are initialized to some value $\tau_i \in \mathcal{R}$. The result of the program is that for some register R_i,*

$$R_i = \tau_i + v$$

If our transparent register program further fulfills the property that

$$R_j = \tau_j \qquad \forall\, j \neq i$$

then we say it is a clean register program. In both cases we say that v is the value that P computes and R_i is the target register, and write

$$P: \quad R_i \leftarrow R_i + v$$

to indicate the function of P. We also occasionally say that P computes v into R_i.

For convenience, we also assume that for every clean register program P there exists a clean register program P^{-1} such that

$$P^{-1}: \quad R_i \leftarrow R_i - v$$

for the same R_i and v as P.

To put this definition to use in the catalytic setting, let us rephrase Lemma 1 in our new language.

Lemma 2. *Let P_1 and P_2 be clean register programs over $\mathbb{F}_2$ computing g_1 and g_2 into target registers R_1 and R_2 respectively. There exist clean register programs*

$$P_\neg: \quad R_0 \leftarrow R_0 + \neg g_1$$

$$P_\wedge : \quad R_0 \leftarrow R_0 + (g_1 \wedge g_2)$$

$P_\neg$ uses only the two registers R_1 and R_0, and makes one recursive call to P_1 plus two basic instructions. $P_\wedge$ uses only the three registers R_1, R_2, and R_0, and makes two recursive calls each to P_1 and P_2 plus four basic instructions.

Moving to the regime of CL allows us to consider a polynomial number of registers, rings of larger size, and so on. Thus we can ask what other functions can be computed by clean register programs using efficient time and space. For example, we could apply the principles of Lemma 1 at a larger scale to handle arbitrarily large products.

Exercise 3. *Let $P_1 \dots P_d$ be clean register programs over $\mathbb{F}_2$, where P_i computes g_i into register R_i for each i. Prove there exists a clean register program*

$$P_\wedge : \quad R_0 \leftarrow R_0 + \wedge_i g_i$$

where $P_\wedge$ uses only the registers $R_1 \dots R_d$, and R_0, and makes $2^{O(d)}$ recursive calls in total plus $2^{O(d)}$ basic instructions.[6]

Focusing on circuit models for which our recursively-structured register programs may be useful, the first stop above L would be AC^1, which contains unbounded fan-in ANDs, or perhaps VP, which contains unbounded fan-in + and fan-in two $\times$ over $\mathbb{Z}$. We will skip ahead to a much greater prize: unbounded fan-in majority.

Theorem 3. $\mathsf{CL} \supseteq \mathsf{TC}^1 (\supseteq \mathsf{NL})$

This statement is clear evidence of the power of catalytic computing; the catalytic tape captures the power of non-determinism, and likely much more.

TC^1 is defined as log depth circuits with unbounded fan-in majority gates, or, equivalently, log depth circuits with unbounded fan-in $g_{=\ell}$ gates for every ℓ, where $g_{=\ell}$ outputs 1 iff the number of 1-inputs is exactly ℓ. This characterization allows us to reduce Theorem 3 to a simple statement about register programs, plus a bit of care in the application.

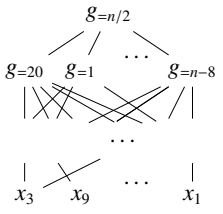


Fig.: TC^1

[6] Note that this can be done with fewer registers and recursive calls by executing Lemma 1 on a tree of fan-in two $\wedge$ gates of height $\log d$, but there is a program that works more directly on the whole product at once, and which is more useful for other applications since it can be generalized.

Lemma 3. *Let $m \in \mathbb{N}$ and let $p > m$ be a sufficiently large prime. For $i \in [m]$, let P_i be a clean register program over $\mathbb{F}_p$ computing*

$$P_i : \quad R_i \leftarrow R_i + v_i$$

Then for every $\ell \in [m]$ there exists a clean register program $P_{=\ell}$ which cleanly computes the indicator function

$$P_{=\ell} : \quad R_0 \leftarrow R_0 + \left[\sum_i v_i = \ell\right]$$

$P_{=\ell}$ uses $O(m)$ registers and makes eight total calls to each P_i plus $O(p)$ basic instructions.

We include an outline of proof below with a number of exercises for readers who want to get a feel for crafting register programs, although readers who want to see the full program for themselves will accordingly find it in the appendix.

Proof. $P_{=\ell}$ works in two parts. First, we define program P_Σ which cleanly computes

$$P_\Sigma : \quad R_\Sigma \leftarrow R_\Sigma + \sum_i v_i$$

given the programs P_i. Second, given a program P_v which cleanly computes some value v into R_v, we define program P_k which cleanly computes

$$P_k : \quad R_0 \leftarrow R_0 + v^k$$

There are two straightforward tasks and one more difficult one. First, construction $P_{=\ell}$ given P_Σ and P_k is almost immediate from Fermat's Little Theorem; since $p > \sum_i v_i$:

$$\left[\sum_i v_i = \ell\right] \equiv 1 - \left(\ell - \sum_i v_i\right)^{p-1} \mod p$$

We define P_v to be the following program:

1. P_Σ^{-1} $\qquad R_\Sigma = [\tau_\Sigma] - \left[\sum_i v_i\right]$

2. $R_\Sigma \mathrel{+}= \ell$ $\qquad R_\Sigma = \left[\tau_\Sigma - \sum_i v_i\right] + [\ell]$
$\qquad\qquad = \tau_\Sigma + \left(\ell - \sum_i v_i\right)$

and thus with P_v being our subroutine to P_k—using $v = \ell - \sum_i v_i$ and $R_v = R_\Sigma$—and choosing $k = p - 1$, the following program computes $P_{=\ell}$:

$$\begin{aligned}
&1.\quad P_{p-1}^{-1} && R_0 &&= [\tau_0] - \left[\left(\ell - \sum_i v_i\right)^{p-1}\right] \\
&2.\quad R_0 \mathrel{+}= 1 && R_0 &&= \left[\tau_0 - \left(\ell - \sum_i v_i\right)^{p-1}\right] + [1] \\
& && &&= \tau_0 + \left(1 - \left(\ell - \sum_i v_i\right)^{p-1}\right) \quad \checkmark
\end{aligned}$$

which completes our program by the previous equation and the fact that our program is over $\mathbb{F}_p$.

Now we need only construct P_Σ and P_k, both of which we leave as exercises. P_Σ is almost immediate and should not be overthought.

Exercise 4. *Construct P_Σ making one call to each P_i and one call to each P_i^{-1}.*

P_k is a bit trickier, but it follows nicely from the following observation:

$$v^k = (\tau_v - (\tau_v - v))^k = \sum_{j=0}^{k} \binom{k}{j} (\tau_v)^j (-1)^{k-j} (\tau_v - v)^{k-j}$$

The utility of this equation is that we always have access to either τ_v (at the start of the program) or $\tau_v - v$ (after running P_v^{-1}). This is a variant of Lemma 1, but it takes a bit of clever thinking, plus using some external memory besides R_v and R_0.

Exercise 5. *Construct P_k making one call each to P_v and P_v^{-1}.*

When all is said and done there is a solution using $m + p + 2 = O(m)$ registers, eight calls to each program P_i or its inverse, and roughly $2p + 12 = O(p)$ basic instructions, although anything in this ballpark works fine. □

We refer to this and similar proofs as *register program arguments*. Such arguments are at the forefront of our knowledge with regards to catalytic computation, as we have not proven a stronger result using compress-or-random up to this point.

3.3 Lower bounds

3.3.1 The average catalytic tape

We began our discussion of the power of catalytic computation with the observation that even an incompressible tape can be useful. We now turn to the other

side of the same coin: the average catalytic tape is incompressible. This has a surprisingly simple implication for the runtime of our machines.

Theorem 4. $\mathsf{CL} \subseteq \mathsf{ZPP}$

If Theorem 3 gives a strong indication that CL is strictly more powerful than L, Theorem 4 is an even stronger signal that CL is exponentially weaker than PSPACE.

Proof. Recall the argument that $\mathsf{L} \subseteq \mathsf{P}$: there are at most $2^{O(\log n)} = \mathsf{poly}\ n$ possible machine configurations, and if any such configuration ever repeats then they must repeat ad infinitum, a contradiction.

Despite having an exponential number of configurations, the same argument applies to CL, albeit only in an average sense. Fix an input x, and consider the configuration graph of our CL machine; rather than a single line of polynomial length emanating from the all zeroes starting node, as in the case of L, there are as many lines as there are starting configurations τ of the catalytic tape, a number which we call m. By extension, the total number of configurations is at most $m \cdot 2^{O(\log n)}$.

One further observation is needed, and it comes from the catalytic restoration property: no two lines coming from different initial catalytic tapes may cross, for if they did then at most one of the two intersecting paths can correctly reset its catalytic tape. Thus the *average* length of a computation path is $\mathsf{poly}\ n$ as before.

Hence we can simulate our CL machine by a randomized P machine by choosing a random catalytic tape and running for a polynomial number of steps. The zero errorness, i.e. inclusion in ZPP rather than BPP, follows because the machine never errs; we declare failure only if the machine takes too long. □

Note that the above argument does not give us a guaranteed poly time bound on the runtime of a catalytic algorithm. Thus it is unknown whether $\mathsf{CL} \subseteq \mathsf{P}$, and Theorem 4 gives us little guidance in solving this problem, as coming up with random strings seems as hard as derandomizing polynomial time classes themselves.

3.3.2 Reversibility

The view of configuration graphs of catalytic machines being collections of lines gives us a nice extension, which is a reversibility property. Reversibility as a technique developed across many papers, most notably those by Bennett [Ben73, Ben89]. We refer the interested reader to the catalytic survey by Koucký [Kou16] for an historic overview.

For our purposes, this is not so much a direct lower bound as a tool used in results similar to Theorem 4, which we will discuss more in the next section. It

also makes formal the intuition that a catalytic machine must undo all its work at the end of a computation. This version of the proof—originally appeared in unpublished work by Dulek [Dul] and later proven by Datta et al. [DGJ+20]—is based on a technique of Lange et al. [LMT00].

Theorem 5. *Let C be a deterministic catalytic machine. Then there is another deterministic catalytic machine C', computing the same function and using the same amount of work and catalytic space as C, such that at any point in the execution of C' on some input x, there is both a unique forward instruction and a unique backward instruction.*

Proof. In the proof of Theorem 4 we noted that after fixing an input x, no two configuration paths coming from different initial catalytic tape configurations can ever meet; hence we called such paths "lines". If the configuration graph was truly a collection of lines then we would be done; at every state there is at most one way forward and one way back.

This view is almost correct, but slightly off: there may be configurations unreachable from any start state but that nevertheless hang off the side of a path; namely, if we were given such an illegal configuration and were asked to take a step forward, we may end up on a legal path. This is no issue when running forwards, but causes some concern when running in reverse.

The solution is to have our new machine C' take an Eulerian tour around the configuration graph of C. Hence as it travels along such forward lines, it may in fact stray into a branch of illegal configurations, but eventually it will work its way around and make it back to the original path. The key point is that it will never reach any state with the wrong answer—we start by altering C to make sure all accept/reject states have outdegree zero—nor any other path coming from a different initial catalytic tape by our earlier discussion.

The details of how to modify the transitions of C to allow this tour to happen, and how to use the right hand rule to avoid infinite loops, are mostly technical curios which we omit. Recall that for our deterministic machine C each transition only relies on a constant amount of information. Some other small tweaks to C may be in order. □

The upshot is that without loss of generality we can assume a catalytic machine runs forward until it discovers the output, then switches direction and runs the same algorithm in reverse until it returns to the beginning.

3.4 Variants of CSPACE

In defining a basic catalytic space-bounded computation class, we invariably have opened the door to a whole parallel complexity hierarchy, both of the basic CSPACE

classes and of augmentations therein. We mention some of these variants and their highlights now.

3.4.1 Choices of s and c

In this survey we mostly focus on the case where $s = \log c$, but this is not the only possible consideration. Bisoyi et al. [BDS22] consider many different regimes, from the *high end* where $s = c^{\epsilon}$, to the *low end* where $s = \log \log c$ or even $s = O(1)$, which they call CR since it corresponds to regular languages with catalycity. Their investigation was preliminary, and so we do not discuss these results in more detail, but the low end regime will come back in some form during our discussion of branching programs in Section 6.

3.4.2 Randomized and non-deterministic computation

Two fundamental resources, both of which we have already seen in the classic space-bounded setting, are randomness and non-determinism. The randomized class CBPL was introduced by Datta et al. [DGJ+20] in relation to Theorem 5, while CNL was defined by Buhrman et al. [BKLS18] in a follow-up to their original work.

In both cases there is an immediate question to be answered: when does the catalytic tape need to be reset? The answer given by both [DGJ+20] in the randomized case and [BKLS18] in the non-deterministic case is the safe one: the catalytic tape must always be reset, whether or not the correct answer is returned.

So far we do not have many results about the additional power of either CBPL or CNL. Structurally we have a few results in the non-deterministic world that mirror the traditional space-bounded setting, such as a (conditional) catalytic Immerman-Szelepcsényi Theorem [Imm88, Sze88], i.e. CNL = coCNL [BKLS18], and a conditional reduction of CNL to its unambiguous variant CUL [GJST19].

One other note about both models is that because we have no explicit runtime restrictions, our catalytic machines are allowed to use a superpolynomial amount of randomness or non-determinism as they so choose. It is then quite surprising that using both our upper bound techniques in tandem, namely average catalytic tapes and reversibility, gives us a way to upper bound both classes in ZPP just as with CL.

Exercise 6. *Use ideas from Theorems 4 and 5 to show that* CBPL *and* CNL *are contained in* ZPP.

Similar arguments can show various other results, such as 1) CBPL is contained in CZPL if we are allowed to read the randomness twice in the latter case; or 2) CL gains no power when we allow it to err during the resetting of the catalytic tape in $O(1)$ spots.

3.4.3 Non-uniform computation

One nice aspect of space complexity is that it is syntactically captured by the branching program model, which in particular gives a straightforward way to think about non-uniform computation. How does this model translate to the catalytic world?

Girard et al. [GKM15] define catalytic branching programs in the following way: rather than having a single start node and going to two different end nodes, we have m different start nodes, each with their own label τ, and $2m$ end nodes which are each labeled with both an output to the function and a start node τ. The catalytic property, naturally, states that on input x, each start node τ must reach exactly the end node labeled with $f(x)$ and τ.

If the program has size $s \cdot m$, we can think of this non-uniformly computing $\mathsf{CSPACE}(\log s, \log m)$: we use $\log sm$ bits to remember the current node, but $\log m$ of which are set at the start and must be reset at the end.

Notice that we no longer need to address into the catalytic tape using the work tape, and so unlike with CSPACE it makes sense to talk about m as being much greater than 2^s. In fact, Potechin [Pot17] showed that for $m = 2^{2^n}$, $s = O(n)$ is sufficient for *any* function f, a non-uniform $\mathsf{CSPACE}(\log n, 2^n) = \mathsf{ALL}$ theorem.

Exercise 7. *Let $P_1 \dots P_n$ be clean register programs where P_i computes x_i into R_i for each i. Show that for any function $f(x_1 \dots x_n)$ there exists a clean register program P_f computing f into R_0, where P_f uses $2^{O(n)}$ registers and makes $O(1)$ recursive calls to each P_i.*

This is optimal in terms of "work space" s, while follow up works of Robere and Zuiddam [RZ21] and Cook and Mertz [CM22] have improved on the "catalytic space" m needed as well.

In the spirit of connecting catalytic computation to broader questions in complexity, we mention that Potechin [Pot17] also made a nice connection of catalytic branching programs to a notion of amortized space-bounded complexity. For a catalytic branching program B with m start nodes of size $s \cdot m$ computing f, we can think of s as being the average size of a branching program needed to compute f, where the averaging is over the m "different branching programs" (one start node, two end nodes) for f embedded inside B. Thus the previous results also show that the amortized branching program size of any function is linear, and the question remains how much amortization is necessary to achieve this.

3.5 Afterword: eyes on the prize

We take a step back once again before moving to the open problems. We have now introduced catalytic computation, with catalytic Turing Machines and a set

of complexity classes, plus some results.

Without any prompting from us, readers to whom this definition appeals can begin to form a network of open problems for themselves. We will leave one last exercise to these readers to start them on their way.

Exercise 8. *Pick your favorite function inside* TC^1—STConn, $\bigoplus$, Tribes,...—*and give a* CL *algorithm for it directly. You can use compress-or-random, register programs, or anything else you like, as long as you use the catalytic tape in an interesting way.*

However, our goal is ultimately to study reusing space, and so we turn back to the reader who is interested in techniques and applications to classical space-bounded classes rather than this new exotic definition. To reiterate: catalytic computation is a test bed for how to use memory both as storage and as computation simultaneously. Thus, for example, when we say

"problem x is computable in catalytic logspace"

we may take this to mean a variety of things aside from what is stated. Practically it could mean

"if we need to compute x as a subroutine many many times to compute problem y, the space to do so need not compound linearly"

or in terms of studying x itself it could mean

"while x may require a lot of space, it can be quite well-structured with regards to its space usage"

et cetera. Similarly, if we show that problem x is in, for example, CBPL or CNL, this means that a space-bounded algorithm with access to randomness or non-determinism can implement space reuse techniques to compute x as above.

Any statement about catalytic computation, even structural results, may be looked at in this way, and we encourage readers to interpret catalytic questions and results in whatever light is most useful.

PART II: WHAT WE DON'T KNOW (YET)

We now move to our second, and main, purpose in this work: a curated list of some of the open questions in the field. Most will rely on terminology introduced in the previous sections, but some will need some definitions later; in either case we endeavor to keep the statements themselves at a high level, and we then expound upon each of them in rough detail in the remainder of the survey.

Below is the complete list of problems, presented in the order in which they will appear. We make no attempt to segregate them by perceived difficulty or concreteness, but we provide this loose table of contents for readers who have a certain type of question in mind, or alternatively for those who want to get a broad sense of what questions are being asked before diving into specifics.

The list of problems

1. Give a simple, direct proof of $\mathsf{uSTConn} \in \mathsf{L}$.
2. Give a simple, direct proof of $\mathsf{uSTConn} \in \mathsf{CL}$.
3. Give a simple, direct proof of $\mathsf{STConn} \in \mathsf{CL}$.
4. Try to improve Savich's Theorem: prove $\mathsf{NSPACE}(s) \subseteq \mathsf{SPACE}(o(s^2))$.
5. Improve the deterministic space complexity of $\mathsf{BPSPACE}(s)$.
6. Decide the space complexity of $\mathsf{TreeEval}$.
7. Give a register program for computing any polynomial $p(x_1 \dots x_n)$ using $O(n)$ registers over a constant size ring $\mathcal{R}$ and $O(1)$ recursive calls P_x, whose effect is to add $x_1 \dots x_n$ to $R_1 \dots R_n$ respectively.
8. Show that for any branching program B of sufficiently large width $w = \Omega(1)$ and length ℓ, there exists a branching program B' of width $w/2$ and length $O(\ell)$ computing the same function.
9. Show that for any branching program B of sufficiently large width w and length ℓ, there exists a branching program B' of width $w - 1$ and length $\mathsf{poly}(\ell)$ computing the same function.
10. Find any function whose optimal space algorithm can be made almost entirely catalytic, i.e. a function requiring—or even that we only know how to do in—$\mathsf{SPACE}(s)$ but which is computable in $\mathsf{CSPACE}(\ll s, \approx s)$.
11. Prove $\mathsf{CL} \subseteq \mathsf{P}$.
12. Show that $\mathsf{L} \subsetneq \mathsf{P}$ implies $\mathsf{CL} \subseteq \mathsf{P}$.
13. Show that $\mathsf{CL} \subseteq \mathsf{P}$ would give strong evidence $\mathsf{ZPP} \subseteq \mathsf{P}$.
14. Show that NC^2, or even any circuit of $\omega(\log n)$ depth, can be computed in CL.

15. Give a register program for computing x^k in the non-commutative setting using linear space and a constant number of recursive calls to x.
16. Show that $\mathsf{BPNC}^1 \subseteq \mathsf{CL}$.
17. Design a catalytic branching program with $2^{O(n)}$ start nodes and total size $2^{O(n)} \cdot O(n)$ for any function f.
18. What is the power of $\mathsf{CL}/poly$, and does it have a natural syntactic characterization?
19. Show the existence of an oracle D such that $\mathsf{CL}^D = \mathsf{EXP}^D$.
20. Extend the $\mathsf{BPL} \subseteq \mathsf{CL}$ simulation to show $\mathsf{CBPL} \subseteq \mathsf{CL}$.
21. Show that CL is equivalent even if we allow $\omega(1)$ many errors on the catalytic tape at the end, or alternatively if we allow $\mathbb{E}_{x,\tau}[O(1)]$ many such errors.
22. Utilize non-determinism in conjunction with catalytic computing in a non-trivial way.
23. Prove $\mathsf{CNSPACE}(s, c) \subseteq \mathsf{CSPACE}(s^2, c^2)$.
24. Implement a catalytic algorithm such that it is actually useful.
25. What does quantum catalytic space look like?
26. Devise a register program using basic instructions inspired by unitary computation, and use it to show non-trivial results for e.g. BQP.
27. Devise a circuit that uses known results from space reuse and catalytic computing to efficiently solve some problem in a way that we do not know how to do directly.
28. Show $\mathsf{TC}^1 \subseteq \mathsf{VP}$.
29. Is the network coding conjecture true or false?
30. Prove or disprove the network coding conjecture when all nodes are restricted to sending linear transformations of their incoming messages.
31. Is there a meaningful notion of a catalytic data structure, or is there anything to be gained from a data structure stored in catalytic memory?
32. Show CL is contained in some subclass of P, perhaps NC, given a believable cryptographic assumption.

33. Show evidence against objects in cryptography based on techniques in reusing space.

34. Show the existence, conditional or otherwise, of a natural class of cryptographic objects by using clean computation.

35. Prove that the existence of one-way functions in CL, or even any one-way function computable by a poly-size poly-length register program, implies the existence of one-way functions in NC^0.

4 What can be done with reusing space?

We first turn our attention to questions in pure (i.e. non-catalytic) space-bounded complexity. Most of these problems will be well-known to the reader, and our only exhortation is to turn the tools seen in this survey upon them for a fresh look.

4.1 Connectivity

Reingold's brilliant result [Rei08] shows that $\mathsf{uSTConn} \in \mathsf{L}$, thus resolving the connection between logspace and symmetric logspace. The algorithm uses tools such as zig-zag product, which are pretty heavy hitting and incur large losses in both time and space. This is in contrast to e.g. the standard RL algorithm, which solves $\mathsf{uSTConn}$ efficiently with very high probability simply by taking a random walk on the graph.

Can reusing space be used to give a simpler, more efficient deterministic algorithm for $\mathsf{uSTConn}$? This would be a useful addition to our study of space-bounded complexity.

Problem 1. *Give a simple, direct proof of* $\mathsf{uSTConn} \in \mathsf{L}$.

Perhaps less ambitiously, consider the case of CL. Both compress-or-random and register programs are enough to show $\mathsf{uSTConn} \in \mathsf{CL}$, but besides being fairly lossy in the constants, as of now neither technique can be nicely described in terms of $\mathsf{uSTConn}$ itself.

A simple, efficient, and clear algorithm for $\mathsf{uSTConn} \in \mathsf{CL}$ would be a useful way of illustrating the counterintuitive power of catalytic space to newcomers in the field, as well as potentially providing an angle on Problem 1.

Problem 2. *Give a simple, direct proof of* $\mathsf{uSTConn} \in \mathsf{CL}$.

We also know that $\mathsf{NL} \subseteq \mathsf{CL}$, meaning we can drop the undirected restriction and still have a CL algorithm for connectivity via register programs. If Problem 2 can be solved, we can hope it also is amenable to such a change.

Problem 3. *Give a simple, direct proof of* $\mathsf{STConn} \in \mathsf{CL}$.

4.2 Savitch's Theorem

Taking a space reuse-style approach to STConn may also give us insights on one of the major unsolved questions in structural space complexity. Savitch's Theorem [Sav70], which states that $\mathsf{NSPACE}(s) \subseteq \mathsf{SPACE}(O(s^2))$, is one of the bedrocks of space complexity, but it is not known to be tight. After over fifty years, it may be time to seriously revisit this question, with reusing space being one of the new tools in our arsenal.

Problem 4. *Try to improve Savitch's Theorem: prove* $\mathsf{NSPACE}(s) \subseteq \mathsf{SPACE}(o(s^2))$.

If Problem 2 gives us a way to solve Problem 1, then perhaps Problem 3 will point the way to attacking Problem 4 in similar fashion.

4.3 Derandomizing space

Derandomizing BPL is a longstanding open problem with a flurry of recent work. There is a wide pool of techniques to draw from, including targeted and weighted PRGs, approximate matrix inversion, certified derandomization, and more; we refer readers to an excellent survey by Hoza [Hoz22] on the topic. Perhaps our techniques will be useful as well.

Problem 5. *Improve the deterministic space complexity of* $\mathsf{BPSPACE}(s)$.

We state Problem 5 less specifically than other problems in this survey because it is an active line of research, and so any target we lay out may be obsolete by the time the reader reaches this survey, and for reasons having nothing to do with reusing space. For example, Hoza [Hoz21] recently improved on the best known upper bound of $\mathsf{SPACE}(s^{3/2})$, due to Saks and Zhou [SZ99], for the first time in thirty years.

4.4 The Tree Evaluation Problem

Another key question in the study of space-bounded computation is how it compares to time-bounded computation. One central question is whether logspace is strictly contained in polynomial time or not.

Cook et al. [CMW+12] proposed that a function known as the Tree Evaluation Problem, or TreeEval for short, may be the key to separating L from P. The function is defined, for an alphabet size k and height h, by a height h rooted full binary tree, where leaves are given values in $[k]$ and internal nodes are labeled

with functions from $[k] \times [k]$ to $[k]$. In other words, it is a sort of alternate circuit model where the values come from a broader alphabet than just $\{0, 1\}$, and the topology is fixed but the functions at each gate are given as input.

Problem 6. *Decide the space complexity of* TreeEval.

The logic for thinking TreeEval $\notin$ L relies exactly on not being able to use space both for memory and for computation. Building on ideas we have seen previously, and in particular a generalization of Lemma 1 first to larger products (see Exercise 3) and then to arbitrary polynomials, Cook and Mertz [CM20, CM21] gave an algorithm for computing TreeEval more efficiently than Cook et al. conjectured. An optimal version of their key register program remains open, and would be sufficient to show TreeEval $\in$ L for all values of k and h.

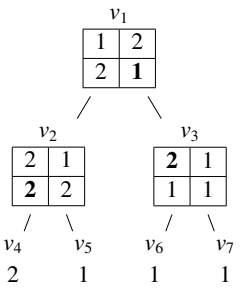


Fig.: TreeEval

Problem 7. *Give a register program for computing any polynomial $p(x_1 \dots x_n)$ using $O(n)$ registers over a constant size ring $\mathcal{R}$ and $O(1)$ recursive calls to program P_x, whose effect is to add $x_1 \dots x_n$ to $R_1 \dots R_n$ respectively.*

4.5 The power of formulas?

Barrington's Theorem allows us to characterize the class NC^1 of all polynomial size formulas as functions computable by branching programs of polynomial length and constant width. This is in contrast to L, whose branching programs can be polynomial in both length and width.

Problem 6 has been posed in the world of formulas as the *KRW conjecture* [KRW95], a depth-based composition theorem that states that no "depth reuse" in the vein of our results should be possible. Resolving the KRW conjecture in the affirmative while also showing TreeEval $\in$ L would separate L from NC^1, an extremely fine-grained separation.

On the other hand, we have seen the surprising power of constant space, and so it is entirely possible that not just L but even NC^1 can implement our techniques. In fact, by the characterizations laid out above, a length-width tradeoff for branching programs would be sufficient.

Problem 8. *Show that for any branching program B of sufficiently large width $w = \Omega(1)$ and length ℓ, there exists a branching program B' of width $w/2$ and length $O(\ell)$ computing the same function.*

Problem 8 would show $\mathsf{NC}^1 = \mathsf{L}$. We note that a Savitch-style argument may work if we relax the length requirement to $\mathsf{poly}(\ell)$, which would only reprove that $\mathsf{L} \subseteq \mathsf{NC}^2$. We formulate an even weaker version of the question just to get the ball rolling.

Problem 9. *Show that for any branching program B of sufficiently large width w and length ℓ, there exists a branching program B' of width $w - 1$ and length* $\mathsf{poly}(\ell)$ *computing the same function.*

In Section 7 we return to the question of other models that may be able to implement our techniques.

5 Where does catalytic fit in to complexity theory?

Moving from catalytic techniques to catalytic computing itself, the most impactful questions remain those relating catalytic computing to more traditional complexity classes. While we already have a reasonably narrow range where classes such as CL can possibly sit, there are many important details to be resolved, as well as orthogonal questions about the use of catalytic computing.

5.1 Space versus catalytic space

At first glance, [BCK+14] settles the question of CSPACE in relation to SPACE once and for all. On one hand, for those that believe that $\mathsf{L} \neq \mathsf{NL}$, there are many natural classes sitting inside $\mathsf{CSPACE}(s, 2^s)$ and outside $\mathsf{SPACE}(s)$. Even more widely believed is that $\mathsf{ZPP} \neq \mathsf{PSPACE}$; in fact, it seems dubious that CL can even contain *any* class $\mathsf{SPACE}(s)$ where $s = \omega(\log n)$, as such classes are widely believed to be separate from P.

Yet if we put aside entire complexity classes for a moment, we can still look to individual problems and ask concrete questions about what catalytic space can offer. Perhaps the most natural such question is to ask an instance-wise version of CL versus PSPACE.

Problem 10. *Find any function whose optimal space algorithm can be made almost entirely catalytic, i.e. a function requiring—or even that we only know how to do in—*$\mathsf{SPACE}(s)$ *but which is computable in* $\mathsf{CSPACE}(\ll s, \approx s)$.

This question, while interesting in its own right, is not a mere curio. Recently Doron and Tell [DT23] gave derandomization with almost no memory overhead conditioned on a few natural assumptions, but to get the optimal result requires a function computable by $\mathsf{CSPACE}(\epsilon s, s)$ but not by $\mathsf{SPACE}((1-\epsilon)s)$; an affirmative answer to our problem would also make this assumption hold unconditionally by

padding. Note that here "almost entirely catalytic" means we cannot tolerate even a factor of 2 in the simulation; this is truly $\mathsf{CSPACE}(s, c)$ versus $\mathsf{SPACE}(s + c)$.

5.2 The catalytic holy grail: CL versus P

Problem 11. *Prove* $\mathsf{CL} \subseteq \mathsf{P}$.

Throughout this survey I have (hopefully without confusion) used hyperbole for stylistic flair, and calling any problem the "holy grail" of a nine-year-old field is as blatant as hyperbole can be. Nevertheless, resolving the relationship between CL and P has proved as fascinating as tenacious, and I can think of no better way to study the structure of catalytic machines than to attempt to resolve Problem 11.

For starters, even a conditional result—short of derandomizing ZPP, of course—could be very useful. It would be interesting to go outside the realm of derandomization in general, or at least to start from an earlier point, such as a novel use of the hardness-versus-randomness paradigm (see e.g. Pyne et al. [PRZ23] for a discussion of such techniques in the space-bounded setting).

Problem 12. *Show that* $\mathsf{L} \subsetneq \mathsf{P}$ *implies* $\mathsf{CL} \subseteq \mathsf{P}$.

On the flip side, there may be barrier results that make the proof difficult even for those who believe $\mathsf{ZPP} = \mathsf{P}$. Buhrman et al. made initial progress on the relativization barrier, showing oracles A and B such that $\mathsf{CL}^A = \mathsf{PSPACE}^A$ and $\mathsf{NL}^B \not\subseteq \mathsf{CL}^B$. Perhaps a more direct barrier result is possible.

Problem 13. *Show that* $\mathsf{CL} \subseteq \mathsf{P}$ *would give strong evidence* $\mathsf{ZPP} \subseteq \mathsf{P}$.

5.3 The power of CL

For those more interested in the power of catalytic computing, the frontier of CL stands at Theorem 3. There is a long way to go even for those who believe $\mathsf{CL} \subseteq \mathsf{P}$.

5.3.1 Going up

The next clear challenge is going beyond logarithmic depth. Consider that when studying alternate problems such as Tree Evaluation, the register program technique has no obvious way of moving beyond objects of logarithmic depth, as recursively computing and uncomputing the inputs to a subroutine leads to runtime costs which are exponential in the recursion depth.

NC circuits contain only fan-in two AND and OR gates—Lemma 1 is meant to handle these circuits specifically—and so looking at $\log^2 n$ depth NC circuits allows us to focus on overcoming the depth barrier. Moving beyond logarithmic

depth in any capacity would then allow us to consider bootstrapping or other such techniques.

Problem 14. *Show that* NC^2*, or even any circuit of* $\omega(\log n)$ *depth, can be computed in* CL.

There is one proposition to Problem 14 that skirts around the difficulty of higher depth: compress many layers into one, and then handle the resulting functions directly using register programs as before. For example, using the fact that NC^1 is contained in L, for which matrix powering is complete, an extension of Lemma 3 to the non-commutative realm could be sufficient to prove $\mathsf{CL} \subseteq \mathsf{NC}^2$.

Problem 15. *Give a register program for computing* x^k *in the non-commutative setting using linear space and a constant number of recursive calls to* x.

To connect this to an earlier problem, all state-of-the-art approaches to Problem 6—i.e. register programs for arbitrary polynomials—work in the non-commutative setting as well, and thus optimal improvements therein may yield non-commutative powering as a special case.

5.3.2 Orthogonal improvements

There are other classes we could study in relation to CL, not linearly up from TC^1 but still interesting. One odd gap in our understanding is that of read-multiple randomness. It is well-known that while $\mathsf{NC}^1 \subseteq \mathsf{L}$, the same is not known for their randomized variants, because BPL has the restriction of only reading its random bits once. This is also the key to using Nisan's argument as it appeared in Theorem 2, and thus the following question is still open.

Problem 16. *Show that* $\mathsf{BPNC}^1 \subseteq \mathsf{CL}$.

If we treat the catalytic tape as a potential source of randomness à la Theorem 2, we can in fact read these "random" bits as many times as we need to compute the circuit. Thus Problem 16 boils down to understanding what properties of randomness are enough to fool circuits, and how to compress when these properties are not met.

5.4 Non-uniform catalytic computation

Moving to the non-uniform setting, we are free from many of the restrictions on CL that we previously faced. While many individual problems can be posed, it seems likely that non-uniform CL is universal even for the strictest setting of parameters, i.e. linear catalytic space and a log space work tape free of constant multipliers. This would obviate most other results that could be proposed.

Problem 17. *Design a catalytic branching program with $2^{O(n)}$ start nodes and total size $2^{O(n)} \cdot O(n)$ for any function f.*

As with Problem 15, this would follow directly from an optimal improvement to Problem 7.

However, we should note that calling this class "non-uniform CL" is somewhat of a misnomer. If we consider the class CL/poly , the equivalence of syntactic models and advice breaks down, as it is not at all obvious that this can capture such a class of exponential-sized branching programs; in fact considering $\mathsf{CL} \subseteq \mathsf{ZPP}$ and $\mathsf{ZPP/poly} = \mathsf{P/poly} \neq \mathsf{ALL}$, these are clearly not equivalent modulo Problem 17. It is worth understanding the actual non-uniform CL in its own right.

Problem 18. *What is the power of* CL/*poly, and does it have a natural syntactic characterization?*

5.5 Oracle results

As mentioned briefly before, Buhrman et al. [BCK+14] also give a few oracle results that complicate our potential attempts to resolve e.g. Problem 11. Oracle results in the world of space-bounded computation are notoriously finicky, and so they may not end up being of much use in the end, but for now it is useful to increase our understanding of catalytic computation.

One result we mentioned previously was an oracle A such that $\mathsf{CL}^A = \mathsf{PSPACE}^A$; the oracle in question is essentially an optimal compress-or-random object, which takes in a string w and either gives a compression of w if it has low entropy or the answer to a PSPACE-complete problem if it has high entropy. This kind of "password oracle" is similar to the one showing $\mathsf{ZPP}^B = \mathsf{EXP}^B$.[7] It seems plausible that the two approaches can be merged into one.

Problem 19. *Show the existence of an oracle D such that $\mathsf{CL}^D = \mathsf{EXP}^D$.*

Note that this would make resolving Problem 11 much more difficult, as it would give an oracle with respect to which $\mathsf{CL}^D \neq \mathsf{P}^D$ by the relativized time hierarchy theorem.

6 Structural catalytic complexity

The second set of questions we have regarding catalytic computing is the structural complexity of catalytic computing itself, as a parallel to the hierarchy of

[7] This is commonly attributed to Heller [Hel86], but according to Morgan Shirley the result as stated is actually unpublished and only appeared during a conference talk for a related paper. Very similar results and proofs do however exist in the literature, and it has been subsumed by Beigel et al. [BBF98]. My thanks to him for finding this information as well as the proof itself.

traditional logspace-bounded classes. Since [BCK+14] introduced catalytic computation, this structural theory has seen the most exposition, and so there are many possible questions to study. We again refer the reader interested in catalytic computation in its own right to the survey of Koucký [Kou16].

6.1 Randomness

6.1.1 Derandomization

Problem 5 posed the question of derandomization for space-bounded complexity classes. We can also ask a catalytic version: if we cannot solve derandomize BPL into L, can we at least do the same with their catalytic variants?

Problem 20. *Extend the* BPL ⊆ CL *simulation to show* CBPL ⊆ CL.

In the catalytic world, we have the compress-or-random argument from Theorem 2 in addition to all the tools coming from the study of BPL. This technique is worth regarding for Problem 20 for at least two reasons: 1) it is likely that variants of the argument can accommodate other derandomization techniques in tandem; and 2) considering CL has a catalytic tape itself, working against BPL with a catalytic tape may not be much harder than working against BPL.

Recall that CBPL ⊆ ZPP, meaning that we can remove the two-sided error from CBPL; in fact the argument only used randomness to pick an initial catalytic tape. [DGJ+20] Thus if CL can find a "good" catalytic tape, we are already done. However, there is an exponentially large configuration graph to consider if we pick the wrong tape. Furthermore our compress-or-random argument now has to work for random tapes that are useful against CBPL rather than CL, a harder condition to quantify.

6.1.2 Robustness

There is a related question, which is the robustness of the catalytic definition to small errors. Every bit of the catalytic tape forgotten is a bit that can be used for free memory, and so we cannot expect CL to remain the same with too many errors, but as of now we only know how to recover from a constant number of errors on the catalytic tape[8], while intuitively even a logarithmic number should hardly be earth-shattering. An alternative result would be to show that CL can recover from only having a constant number of errors on average, rather than for every input and catalytic tape.

[8]This result is originally due to Jain et al. and will appear in upcoming work. The proof can once again be seen by taking an Eulerian tour around the configuration graph, while keeping track of the (polynomially bounded, since there are only $O(1)$ errors possible on the catalytic tape) number of starting configurations we pass so we can find the correct one to go back to.

Problem 21. *Show that* CL *is equivalent even if we allow* $\omega(1)$ *many errors on the catalytic tape at the end, or alternatively if we allow* $\mathbb{E}_{x,\tau}[O(1)]$ *many such errors.*

6.2 Non-determinism

Despite our structural results in the non-deterministic setting, we have no natural problems which have been placed in CNL but that are not known to be in CL. Right now non-deterministic catalytic computation seems to be a definition in search of an application.

Problem 22. *Utilize non-determinism in conjunction with catalytic computing in a non-trivial way.*

As a sign of our paucity of knowledge with regards to CNL, we do not yet have an equivalent of Savitch's Theorem in the catalytic world. This is one of the most fundamental questions remaining in importing the structural theory space complexity to the catalytic computing world.

Problem 23. *Prove* $\mathsf{CNSPACE}(s, c) \subseteq \mathsf{CSPACE}(s^2, c^2)$.

As in Problem 20, the tableau method of Savitch's original proof will result in an exponential number of configurations being considered, which seems to be the major obstacle. Even putting this aside, however, such a tableau of configurations of the original CNSPACE machine would have to be kept on the catalytic tape of the simulating CSPACE machine, to be written and read off without issue.

With that said, we note that $\mathsf{CNSPACE}(s, c)$ has the same $\mathsf{ZPTIME}(2^s)$ upper bound as $\mathsf{CSPACE}(s, c)$ [DGJ+20], and there has been no work to suggest it is significantly stronger.

7 Reusing space beyond space

Our last set of problems is more speculative than the rest. In a word, we ask where techniques involving space reuse may find traction outside the study of space-bounded complexity itself.

There are as many proposals to do so as there are fields, and throughout this survey the reader has been encouraged to think to their own research, of which the author certainly knows very little. These are just the ones I personally see as having potential.

7.1 Practical implementations

When Buhrman et al. introduced catalytic computation, they began with a fun illustration of the potential of the model:

> Imagine the following scenario. You want to perform a computation that requires more memory than you currently have available on your computer. One way of dealing with this problem is by installing a new hard drive. As it turns out you have a hard drive but it is full with data, pictures, movies, files, etc. You don't need to access that data at the moment but you also don't want to erase it. Can you use the hard drive for your computation, possibly altering its contents temporarily, guaranteeing that when the computation is completed, the hard drive is back in its original state with all the data intact? [BCK+14]

Before considering any theoretical models beyond Turing machines, we should see if the algorithms we already have in the abstract can be put into concrete use.

Problem 24. *Implement a catalytic algorithm such that it is actually useful.*

James Cook [Coo21] has already taken a stab at streamlining the program in Theorem 3 for STConn and seeing how it does on an actual computer; his program involves concrete tricks from programming, such as replacing addition with rotation. If we take seriously the promise of using hard disk space as our catalytic memory, there are many optimizations that would need to happen in order to make such algorithms practical.

7.2 Quantum computation

As stated in Theorem 5, catalytic computation is built from reversible operations. This calls to mind another complexity setting where every operation is invertible: evolution by unitaries, i.e. quantum computation.

There are many potential ways of approaching the potential connection between quantum and catalytic computation. One is to observe that the one-clean qubit model, known as DQC1 [KL98], is a catalytic-looking quantum model that has been widely studied for many years. Another is to think about how efficient space simulation of circuits may carry over when intermediate circuit computations no longer have the locality we exploited to get down to constant space in Section 2. A third is more optimistic: what other reversible tools for space reuse can come from the wide toolbox of unitaries?

Before jumping the gun, the precise definition of quantum catalytic computing has to be nailed down.

Problem 25. *What does quantum catalytic space look like?*

Even more so than with CBPL and CNL, care needs to be taken to get the right definition. Probably the most standard model in quantum computation is that of quantum circuits, capturing such classes as BQP. However, a circuit has a fixed notion of time, i.e. depth, and as we saw in Theorem 4, while we can reason about the average time a catalytic algorithm could take, there is no subexponential guarantee on the worst-case runtime of algorithms such as the compress-or-random argument in Theorem 2. Thus the quantum Turing Machine model of Watrous [Wat99], while somewhat non-standard, may be more appropriate.

Instead of focusing on catalytic computing per se, we could also look to existing models of quantum computation and ask what techniques such as register programs could buy us.

Problem 26. *Devise a register program using basic instructions inspired by unitary computation, and use it to show non-trivial results for e.g.* BQP.

7.3 Circuits

Recall that our main examples of space reuse, such as our "theorem" in Section 2, were about space-efficient computation of circuits. In the hierarchy of syntactic models, it is known that circuits can efficiently simulate branching programs, while there are separations known in the other direction; hence why results such as Theorem 3 are so interesting.

However, if we consider the power of circuits to simulate branching programs, this means that in some ways a circuit class of sufficient power should be able to implement the space reuse techniques we have seen. It is unclear what that actually looks like when we convert our branching programs over, or if it is even necessary or interesting to do so.

Problem 27. *Devise a circuit that uses techniques from space reuse and catalytic computing to solve some problem in low size or depth in a way that we do not know how to do directly.*

With all the circuit classes sitting between L and P, there is also the question of whether or not implementing catalytic-style techniques can help clean up this landscape.

For example, a longstanding conjecture of Immerman and Landau [IL95] states that the determinant of integral matrices is complete for TC^1. It is known that the determinant is complete for the class of log-depth arithmetic circuits whose + gates have unbounded fan-in and whose $\times$ gates have fan-in two, a class known as VP. Given that Theorem 3 states that TC^1 can be simulated by register

programs, and such programs are arithmetic in nature, perhaps VP can implement our theorem as well.

Problem 28. *Show* $\mathsf{TC}^1 \subseteq \mathsf{VP}$.

7.4 Network coding

Our first example of saving space in this survey was the XOR trick for swapping values. Between this and our register program technique, it makes sense to look at areas where bit tricks and XOR tricks come in handy.

One such area is the network coding conjecture. In essence, the question is whether or not, for a given undirected graph with capacities on each edge and a set of source-sink pairs, we can send more message bits from each source to its corresponding sink than the network flow should allow if we think of the messages as being generic commodities instead. While this seems implausible, we note that this is in fact possible in the directed graph setting, as alluded to in our bulleted list in Section 1, and the counterexample uses an XOR trick.

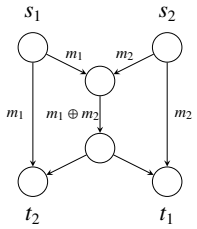


Fig.: network coding (directed counterexample)

Problem 29. *Is the network coding conjecture true or false?*

Resolving Problem 29 in the affirmative, i.e. showing that no tricks are possible for sending more bits than the network flow, would lead to circuit lower bounds for a number of fundamental problems such as sorting [FHLS20] and multiplication [AFKL19].

There are many potential approaches to Problem 29 beyond XOR tricks, but for our purposes it is sufficient to focus on this case for starters. Such "linear" strategies are known to not be optimal in terms of how many bits can be sent [DFZ05], but to the best of our knowledge we still have not resolved the network coding conjecture even in this case.

Problem 30. *Prove or disprove the network coding conjecture when all nodes are restricted to sending linear transformations of their incoming messages.*

7.5 Data structures

One setting where space usage is a key metric to study is that of data structures. Unlike in the usual complexity setting, the balance of time and space is separated into two phases: first, a preprocessing phase where some amount of space is

consumed in order to prepare useful information about the input; and second, we receive a list of queries about the input that we want to answer quickly, using both the input itself as well as whatever we prepared in the first phase.

This two-phase process somewhat muddies the waters when it comes to thinking about space reuse, which is all about recomputing things many times, usually at the cost of time, in order to avoid some bottleneck in the information we have to store. It seems to be an orthogonal style of question.

Nevertheless, with space as one of the main players in the world of data structures, it seems that at least a basic investigation may be warranted.

Problem 31. *Is there a meaningful notion of a catalytic data structure, or is there anything to be gained from a data structure stored in catalytic memory?*

7.6 Cryptography

As with quantum computing, there are many ways to approach the possible link between reusing space and cryptography.

An obvious consequence of Theorem 4, as well as other explicit results [BKLS18, GJST19], is that cryptography implies lower bounds against catalytic computing, as an appropriate pseudorandom generator would show CL and many of its variants are contained in P. Such results could also resolve many of our questions about randomized computation without using any of our space reuse techniques; from the perspective of this survey this would be somewhat disappointing, but in the end results are results.

Problem 32. *Show* CL *is contained in some subclass of* P*, perhaps* NC*, given a believable cryptographic assumption.*

There is also the contrapositive side to Problem 32, of whether reusing space provides a potential barrier against cryptography just as it was a barrier against composition for space. For example, the notion of proofs of space, a variant of the proofs of work paradigm used in cryptocurrency, was complicated by Pietrzak [Pie19], who showed that a stronger object known as catalytic proofs of space would be necessary to constitute an actual "proof of work".

Problem 33. *Show evidence against objects in cryptography based on techniques in reusing space.*

To again err on the side of optimism however, we prefer to ask not what cryptography can do for catalytic computing, but rather what catalytic computing can do for cryptography.

While compress-or-random seems like the more relevant argument for randomized computation, we would like to draw attention to the register program

argument at this juncture. Consider our proof of e.g. Lemma 1 again, and now let our initial values τ_i be chosen i.i.d. from $\{0, 1\}$. Then a clear consequence is that at every individual point in the execution of our algorithm, our memory is distributed i.i.d. as well, i.e. (τ_0, τ_1, τ_2) is distributed uniformly across $\{0, 1\}^3$.

In other words, our memory at every point in time reveals no secrets about the computation of f. This calls to mind potential applications such as homomorphic encryption, leakage resilience, etc. There are definitions and problems to be codified, and more steps may be needed for this approach to work; for example, any *two* points in time together may tell quite a bit about the computation of f.

Problem 34. *Show the existence, conditional or otherwise, of a natural class of cryptographic objects by using clean computation.*

A more concrete problem would be to extend the results of Applebaum et al. [AIK06] to show that cryptography in CL implies cryptography in NC^0. The warm-up for their result is based on Barrington's Theorem, i.e. our "theorem" from Section 2, and so it seems natural to believe that register programs in general could be turned into one-way functions with appropriate tinkering.

Problem 35. *Prove that the existence of one-way functions in* CL*, or even any one-way function computable by a poly-size poly-length register program, implies the existence of one-way functions in* NC^0.

8 Conclusion: to a broader theory

This brings an end to our list of open questions. We will conclude this survey, just as we began, with an entreaty, and my gratitude, to the reader who has made it to the end.

The two most important questions in the field of reusing space are also the most general: to develop new techniques and to develop new applications. New techniques may involve using more cutting-edge tools in complexity, group theory, etc., or they may simply be new ways of approaching the model beyond compression or arithmetic-style reuse. New applications could be with questions that have to do with space, or it could be that the techniques we have described in this work have a home in a very different model.

This survey was an attempt to solve a third, no less important problem: to develop new interest. I hope to have provided the reader with enough of a peek at this burgeoning field to garner interest, and to have provided at least a few problems which can be played with without too much further context. With any luck these techniques and problems will be only a prelude to the exploration of a wider world of reusing space.

References

[AFKL19] Peyman Afshani, Casper Benjamin Freksen, Lior Kamma, and Kasper Green Larsen. Lower bounds for multiplication via network coding. In *International Colloquium on Automata, Languages, and Programming, ICALP*, volume 132 of *LIPIcs*, pages 10:1–10:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in nc^0. *SIAM J. Comput.*, 36(4):845–888, 2006.

[Bar89] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

[BBF98] Richard Beigel, Harry Buhrman, and Lance Fortnow. NP might not be as easy as detecting unique solutions. In *ACM Symposium on the Theory of Computing (STOC)*, pages 203–208. ACM, 1998.

[BC92] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM J. Comput.*, 21(1):54–58, 1992.

[BCK+14] Harry Buhrman, Richard Cleve, Michal Koucký, Bruno Loff, and Florian Speelman. Computing with a full memory: catalytic space. In *Symposium on Theory of Computing, STOC 2014*, pages 857–866. ACM, 2014.

[BDS22] Sagar Bisoyi, Krishnamoorthy Dinesh, and Jayalal Sarma. On pure space vs catalytic space. *Theor. Comput. Sci.*, 921:112–126, 2022.

[Ben73] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, 1973.

[Ben89] Charles H. Bennett. Time/space trade-offs for reversible computation. *SIAM J. Comput.*, 18(4):766–776, 1989.

[BKLS18] Harry Buhrman, Michal Koucký, Bruno Loff, and Florian Speelman. Catalytic space: Non-determinism and hierarchy. *Theory Comput. Syst.*, 62(1):116–135, 2018.

[CM20] James Cook and Ian Mertz. Catalytic approaches to the tree evaluation problem. In *Proceedings of the 52nd Annual ACM Symposium on Theory of Computing, STOC 2020*, pages 752–760. ACM, 2020.

[CM21] James Cook and Ian Mertz. Encodings and the tree evaluation problem. *Electron. Colloquium Comput. Complex.*, page 54, 2021. URL: https://eccc.weizmann.ac.il/report/2021/054.

[CM22] James Cook and Ian Mertz. Trading time and space in catalytic branching programs. In *37th Computational Complexity Conference, CCC 2022*, volume 234 of *LIPIcs*, pages 8:1–8:21, 2022.

[CMW+12] Stephen A. Cook, Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam. Pebbles and branching programs for tree evaluation. *ACM Trans. Comput. Theory*, 3(2):4:1–4:43, 2012.

[Coo21] James Cook. How to borrow memory, 2021. URL: https://www.falsifian.org/blog/2021/06/04/catalytic/.

[DFZ05] R. Dougherty, C. Freiling, and K. Zeger. Insufficiency of linear coding in network information flow. *IEEE Transactions on Information Theory*, 51(8):2745–2759, 2005.

[DGJ+20] Samir Datta, Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. Randomized and symmetric catalytic computation. In *CSR*, volume 12159 of *Lecture Notes in Computer Science*, pages 211–223. Springer, 2020.

[DT23] Dean Doron and Roei Tell. Derandomization with minimal memory footprint. In Amnon Ta-Shma, editor, *Computational Complexity Conference, CCC 2023*, volume 264 of *LIPIcs*, pages 11:1–11:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[Dul] Yfke Dulek. Catalytic space: on reversibility and multiple-access randomness.

[FHLS20] Alireza Farhadi, Mohammad Taghi Hajiaghayi, Kasper Green Larsen, and Elaine Shi. Lower bounds for external memory integer sorting via network coding. *Commun. ACM*, 63(10):97–105, 2020.

[GJST19] Chetan Gupta, Rahul Jain, Vimal Raj Sharma, and Raghunath Tewari. Unambiguous catalytic computation. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019*, volume 150 of *LIPIcs*, pages 16:1–16:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[GKM15] Vincent Girard, Michal Koucky, and Pierre McKenzie. Nonuniform catalytic space and the direct sum for space. *Electronic Colloquium on Computational Complexity (ECCC)*, 138, 2015.

[Hel86] Hans Heller. On relativized exponential and probabilistic complexity classes. *Inf. Control.*, 71(3):231–243, 1986.

[Hoz21] William M. Hoza. Better pseudodistributions and derandomization for space-bounded computation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021*, volume 207 of *LIPIcs*, pages 28:1–28:23, 2021.

[Hoz22] William M. Hoza. Recent progress on derandomizing space-bounded computation. *Bull. EATCS*, 138, 2022.

[IL95] Neil Immerman and Susan Landau. The complexity of iterated multiplication. *Inf. Comput.*, 116(1):103–116, 1995.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988.

[KL98] E. Knill and R. Laflamme. Power of one bit of quantum information. *Phys. Rev. Lett.*, 81:5672–5675, 1998.

[Kou16] Michal Koucký. Catalytic computation. *Bull. EATCS*, 118, 2016.

[KRW95] Mauricio Karchmer, Ran Raz, and Avi Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Comput. Complex.*, 5(3/4):191–204, 1995.

[LMT00] Klaus-Jörn Lange, Pierre McKenzie, and Alain Tapp. Reversible space equals deterministic space. *J. Comput. Syst. Sci.*, 60(2):354–367, 2000.

[Lof] Bruno Loff. Private correspondence.

[Nis93] Noam Nisan. On read-once vs. multiple access to randomness in logspace. *Theor. Comput. Sci.*, 107(1):135–144, 1993.

[Pie19] Krzysztof Pietrzak. Proofs of catalytic space. In Avrim Blum, editor, *Innovations in Theoretical Computer Science Conference, ITCS*, volume 124 of *LIPIcs*, pages 59:1–59:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[Pot17] Aaron Potechin. A note on amortized branching program complexity. In *Computational Complexity Conference*, volume 79 of *LIPIcs*, pages 4:1–4:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[PRZ23] Edward Pyne, Ran Raz, and Wei Zhan. Certified hardness vs. randomness for log-space. *CoRR*, 2023.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, 2008.

[RZ21] Robert Robere and Jeroen Zuiddam. Amortized circuit complexity, formal complexity measures, and catalytic algorithms. In *FOCS*, pages 759–769. IEEE, 2021.

[Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[SZ99] Michael E. Saks and Shiyu Zhou. BP $_{\mathrm{h}}$space(s) subseteq dspace($s^{3/2}$). *J. Comput. Syst. Sci.*, 58(2):376–403, 1999.

[Sze88] Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[Wat99] John Watrous. Space-bounded quantum complexity. *J. Comput. Syst. Sci.*, 59(2):281–326, 1999.

Exercise solutions

Solution 1. *As stated, we will only seek to reprove the correctness of the program for $g_1 \wedge g_2$. Define $y_1 = \tau_1 + g_1$ and $y_2 = \tau_2 + g_2$; in other words, before running P_1 the register R_1 contains τ_1, while afterwards it contains y_1, and likewise for P_2. Then by simple arithmetic*

$$g_1 g_2 \equiv y_1 y_2 + y_1 \tau_2 + \tau_1 y_2 + \tau_1 \tau_2 \mod 2$$

Thus we can add $g_1 g_2$ to R_0 by adding each of the four monomials in this expansion to R_0, one by one, each obtained by running a combination of P_1 and P_2.

In fact, by ordering them as

$$g_1 g_2 \equiv \tau_1 \tau_2 + \tau_1 y_2 + y_1 y_2 + y_1 \tau_2 \mod 2$$

this can be done using only four recursive calls as presented in the original proof of Lemma 1, rather than however many were in our subsequent exposition.

Solution 2. *We will use the same recursive statement as Lemma 1, with two changes. First, we will need a program to handle gates of the form $g = g_1 + g_2$. This is clearly accomplished by the following program $P_g(i)$:*

1.	$P_1(i)$	R_i	$= [\tau_i] + [v_1]$
2.	$P_2(i)$	R_i	$= [\tau_i + v_1] + [v_2] = \tau_i + v_1 + v_2$ ✓

Before moving on to $\times$ gates, we need to handle the fact that executing $P_g(i)$ twice no longer resets memory, as we are no longer over $\mathbb{F}_2$. To handle this, we will extend our recursion to state that in addition to $P_g(i)$ we also have a program $P_g^{-1}(i)$ whose function is to subtract v_g from R_i. This is clearly true at the leaves since reading inputs is atomic, and swapping $P_1(i)$ and $P_2(i)$ for their inverses in our + program above gives $P_g^{-1}(i)$.

Now we can move on to $g = g_1 g_2$. Our program from Lemma 1 works almost directly for $\times$ gates, with the only catch being that we convert + into − in some places in order to get the cancellations to work. Concretely our program $P_g(0)$ is as follows:

1.	$P_1(1)$	R_1	$= [\tau_1] + [v_1]$
2.	$R_0 \mathrel{+}= -R_1 R_2$	R_0	$= [\tau_0] + [-(\tau_1 + v_1)(\tau_2)]$
			$= \tau_0 - \tau_1\tau_2 - v_1\tau_2$
3.	$P_2(2)$	R_2	$= [\tau_2] + [v_2]$
4.	$R_0 \mathrel{+}= R_1 R_2$	R_0	$= [\tau_0 - \tau_1\tau_2 - v_1\tau_2] + [(\tau_1 + v_1)(\tau_2 + v_2)]$
			$= \tau_0 + \tau_1 v_2 + v_1 v_2$
5.	$P_1^{-1}(1)$	R_1	$= [\tau_1 + v_1] + [-v_1]$
			$= \tau_1$ ✓
6.	$R_0 \mathrel{+}= -R_1 R_2$	R_0	$= [\tau_0 + \tau_1 v_2 + v_1 v_2] + [-(\tau_1)(\tau_2 + v_2)]$
			$= \tau_0 - \tau_1\tau_2 + v_1 v_2$
7.	$P_2^{-1}(2)$	R_2	$= [\tau_2 + v_2] + [-v_2]$
			$= \tau_2$ ✓
8.	$R_0 \mathrel{+}= R_1 R_2$	R_0	$= [\tau_0 - \tau_1\tau_2 + v_1 v_2] + [(\tau_1)(\tau_2)]$
			$= \tau_0 + v_1 v_2$ ✓

and it is easy to show that running the same program with all signs flipped gives $P_g^{-1}(0)$ as well. Once again all other programs $P_g^{-1}(i)$ can be obtained by relabeling.

Solution 3. *The following program computes $P_\wedge$:*

for all $S \subseteq [d]$:

1. $P_i \qquad \forall i \in S$
2. $R_0 \mathrel{+}= \prod_{i=1}^{d} R_i$
3. $P_i^{-1} \qquad \forall i \in S$

This follows by a generalization of the alternate analysis given in the previous exercise, namely

$$\prod_{i=1}^{d} g_i \equiv \sum_{S \subseteq [d]} \left(\prod_{i \in S} \tau_i + x_i \right) \left(\prod_{i \notin S} \tau_i \right)$$

by inclusion-exclusion.

Solution 4. *The following program computes P_Σ:*

1. $P_i \qquad \forall i$ $\qquad R_i = [\tau_i] + [v_i] \qquad \forall i$
2. $R_\Sigma \mathrel{+}= \sum_i R_i$ $\qquad R_0 = [\tau_0] + \left[\sum_i (\tau_i + v_i) \right]$
   $= \tau_0 + \sum_i \tau_i + \sum_i v_i$
3. $P_i^{-1} \qquad \forall i$ $\qquad R_i = [\tau_i + v_i] + [-v_i] \qquad \forall i$
   $= \tau_i \quad \forall i \quad \checkmark$
4. $R_\Sigma \mathrel{+}= -\sum_i R_i$ $\qquad R_0 = \left[\tau_0 + \sum_i \tau_i + \sum_i v_i \right] + \left[-\sum_i \tau_i \right]$
   $= \tau_0 + \sum_i v_i \quad \checkmark$

We note that this also gives a program P_Σ^{-1} by running the program in reverse order and flipping all signs.

Solution 5. *The following program computes P_k given P_v, and uses k registers $R'_1 \ldots R'_k$ in addition to R_0 and R_v:*

1. P_v^{-1} $\qquad R_v = [\tau_v] + [-v]$
2. $R'_j \mathrel{+}= (R_v)^j \qquad \forall j = 0...k$ $\qquad R'_j = [\tau'_j] + [(\tau_v - v)^j] \qquad \forall j = 0...k$
3. P_v $\qquad R_v = [\tau_v - v] + [v]$
   $= \tau_v$
4. $R_0 \mathrel{+}= \sum_{j=0}^{k} \binom{k}{j} (R_v)^j (-1)^{k-j} R'_{k-j}$ $\qquad R_0 = [\tau_0] + \left[\sum_{j=0}^{k} \binom{k}{j} \tau_v^j (-1)^{k-j} (\tau'_{k-j} + (\tau_v - v)^{k-j}) \right]$
   $= \tau_0 + \sum_{j=0}^{k} \binom{k}{j} \tau_v^j (-1)^{k-j} (\tau'_{k-j})$
   $+ \sum_{j=0}^{k} \binom{k}{j} \tau_v^j (-1)^{k-j} (\tau_v - v)^{k-j}$

5.	P_v^{-1}	R_v	$= [\tau_v] + [-v]$
6.	$R'_j \mathrel{+}= -(R_v)^j \qquad \forall j = 0...k$	R'_j	$= [\tau'_j + (\tau_v - v)^j] + [-(\tau_v - v)^j]$ $= \tau'_j \qquad \forall j = 0...k \quad \checkmark$
7.	P_v	R_v	$= [\tau_v - v] + [v]$ $= \tau_v \quad \checkmark$
8.	$R_0 \mathrel{+}= -\sum_{j=0}^{k} \binom{k}{j} (R_v)^j (-1)^{k-j} R'_{k-j}$	R_0	$= [\tau_0 + \sum_{j=0}^{k} \binom{k}{j} \tau_v^j (-1)^{k-j} (\tau'_{k-j})$ $+ \sum_{j=0}^{k} \binom{k}{j} \tau_v^j (-1)^{k-j} (\tau_v - v)^{k-j}]$ $+ \left[- \sum_{j=0}^{k} \binom{k}{j} \tau_v^j (-1)^{k-j} (\tau'_{k-j}) \right]$ $= \tau_0 + \sum_{j=0}^{k} \binom{k}{j} \tau_v^j (-1)^{k-j} (\tau_v - v)^{k-j}$ $= \tau_0 + v^k \quad \checkmark$

As before we get an inverse program by running in reverse and flipping all signs.

Solution 6. *The same averaging argument as Theorem 4 shows that a random catalytic tape will only have a polynomial size component in the configuration graph, even when this component is no longer a line and can branch based on randomness or non-uniformity. We choose a random catalytic tape using the randomness of* ZPP*, copy it to remember where we started, and then take an Eulerian tour of the given component in the same way as Theorem 5.*

To simulate CNL *it is enough to look for any accept state we see before returing to our starting configuration, while for* CBPL *we keep a list of how many accept and reject states we encounter and take a majority vote. Note that in either case we will never get the wrong answer, although as before we may have to declare "I don't know" if we pick a bad starting catalytic tape and the procedure takes too long.*

Solution 7. *As in our alternate proof of Lemma 1, define $y_i = \tau_i + x_i$ for each i. Let $p_f(x_1 \dots x_n)$ be the $\mathbb{F}_2$ polynomial computing f given by interpolation, and let $q_f(\tau_1 \dots \tau_n, y_1 \dots y_n)$ be defined by substitution into p_f, namely*

$$q_f(\tau_1, \dots, \tau_n, y_1, \dots, y_n) = p_f(y_1 - \tau_1, \dots, y_n - \tau_n)$$

Given this substitution, consider all monomials in q_f, i.e.

$$q_f = \sum_{S,T \subseteq [n]} c_{S,T} (\prod_{i \in S} \tau_i)(\prod_{i \in T} y_i)$$

for some constants $c_{S,T} \in \mathbb{F}_2$. Note that by construction we need only consider disjoint S and T, since p_f is over $\mathbb{F}_2$ and thus is multilinear. For convenience we relabel these products τ^S and y^T respectively.

We will use registers R_i for each i as well as registers $R_{S,\tau}$ and $R_{T,y}$ for each $S, T \subseteq [n]$. Our idea is to store each τ^S into the corresponding $R_{S,\tau}$ and similarly for y, so that

$$R_{S,\tau} \cdot R_{T,y} = (\tau_{S,\tau} + \tau^S)(\tau_{T,y} + y^T) = \tau_{S,\tau}\tau_{T,y} + \tau_{S,\tau} y^T + \tau^S \tau_{T,y} + \tau^S y^T$$

and as in Lemma 1 we use four rounds to cancel out the various junk terms to get exactly $\tau^S y^T$. Thus the following program computes P_f:

1. $R_{S,\tau}$ += $\prod_{i\in S} R_i$ $\quad \forall S \subseteq [n]$	$R_{S,\tau}$	$= [\tau_{S,\tau}] + [\tau^S] \quad \forall S \subseteq [n]$
2. R_0 += $\sum_{S,T\subseteq[n]} c_{S,T} R_{S,\tau} R_{T,y}$	R_0	$= [\tau_0] + \left[\sum_{S,T\subseteq[n]} c_{S,T}(\tau_{S,\tau} + \tau^S)(\tau_{T,y})\right]$
		$= \tau_0 + \sum_{S,T\subseteq[n]} c_{S,T}(\tau_{S,\tau}\tau_{T,y} + \tau^S \tau_{T,y})$
3. P_i $\quad \forall i \in [n]$	R_i	$= [\tau_i] + [v_i] \quad \forall i \in [n]$
4. $R_{T,y}$ += $\prod_{i\in T} R_i$ $\quad \forall T \subseteq [n]$	$R_{T,y}$	$= [\tau_{T,y}] + [y^T] \quad \forall T \subseteq [n]$
5. R_0 += $\sum_{S,T\subseteq[n]} c_{S,T} R_{S,\tau} R_{T,y}$	R_0	$= [\tau_0 + \sum_{S,T\subseteq[n]} c_{S,T}(\tau_{S,\tau}\tau_{T,y} + \tau^S \tau_{T,y})]$
		$+ \left[\sum_{S,T\subseteq[n]} c_{S,T}(\tau_{S,\tau} + \tau^S)(\tau_{T,y} + y^T)\right]$
		$= \tau_0 + \sum_{S,T\subseteq[n]} c_{S,T}(\tau_{S,\tau} y^T + \tau^S y^T)$
6. $R_{S,\tau}$ += $\prod_{i\in S} R_i$ $\quad \forall S \subseteq [n]$	$R_{S,\tau}$	$= [\tau_{S,\tau} + \tau^S] + [\tau^S] \quad \forall S \subseteq [n]$
		$= \tau_{S,\tau} \quad \forall S \subseteq [n] \quad \checkmark$
7. R_0 += + $\sum_{S,T\subseteq[n]} c_{S,T} R_{S,\tau} R_{T,y}$	R_0	$= [\tau_0 + \sum_{S,T\subseteq[n]} c_{S,T}(\tau_{S,\tau} y^T + \tau^S y^T)]$
		$+ \left[\sum_{S,T\subseteq[n]} c_{S,T}\tau_{S,\tau}(\tau_{T,y} + y^T)\right]$
		$= \tau_0 + \sum_{S,T\subseteq[n]} c_{S,T}(\tau_{S,\tau}\tau_{T,y} + \tau^S y^T)$
8. P_i $\quad \forall i \in [n]$	R_i	$= [\tau_i + v_i] + [v_i] \quad \forall i \in [n]$
		$= \tau_i \quad \forall i \in [n] \quad \checkmark$
9. $R_{T,y}$ += + $\prod_{i\in T} R_i$ $\quad \forall T \subseteq [n]$	$R_{T,y}$	$= [\tau_{T,y} + y^T] + [y^T] \quad \forall T \subseteq [n]$
		$= \tau_{T,y} \quad \forall T \subseteq [n] \quad \checkmark$
10. R_0 += + $\sum_{S,T\subseteq[n]} c_{S,T} R_{S,\tau} R_{T,y}$	R_0	$= [\tau_0 + \sum_{S,T\subseteq[n]} c_{S,T}(\tau_{S,\tau}\tau_{T,y} + \tau^S y^T)]$
		$+ \left[\sum_{S,T\subseteq[n]} c_{S,T}\tau_{S,\tau}\tau_{T,y}\right]$
		$= \tau_0 + \sum_{S,T\subseteq[n]} c_{S,T}\tau^S y^T = \tau_0 + q_f \quad \checkmark$

Solution 8. *The world is your oyster.*

The Distributed Computing Column

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

This month, the Distributed Computing Column is featuring Dean Leitersdorf, winner of the 2023 Principles of Distributed Computing Doctoral Dissertation Award. His work on sparse matrix multiplication has led to several breakthroughs that improve significantly on the state of the art. These new approaches have led to faster algorithms for a variety of related problems, including constant-round algorithms for computing graph spanners, approximate all-pairs-shortest-paths, and the girth of a graph (up to an additive 1) in the congested clique model.

Recent progress in distributed matrix multiplication has been fast, and real-world applications of matrix multiplication have only been increasing in importance. In this column, Dean Leitersdorf gives an overview of the state of the art for distributed matrix multiplication, and its connection to all-pairs shortest paths in the congested clique model. He provides both a summary of his new sparity-aware approach, along with a discussion of open questions and possible future directions. Overall, then, this column provides a succinct overview of a rapidly moving area of distributed algorithms!

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 141

The Relationship between APSP and Matrix Multiplication in Congested Clique

Dean Leitersdorf

Introduction

The field of distributed computing has recently explored the fundamental bidirectional relationship between matrix multiplication and the all-pairs-shortest-path (APSP) problem in the distributed Congested Clique model. Matrix multiplication is the foundation for a wide variety of problems in the exact sciences, with significant algorithmic research effort having been invested in finding the smallest ω such that sequential matrix multiplication [12, 20, 33, 34] can be computed in $O(n^\omega)$ time (where $\omega < 2.37286$ is the best currently-known result [4]). Similarly, graph theory has been the bedrock of computer science research for much of the past century, with the fundamental problem of distance computation having wide implications. While it is well known that the all-pairs-shortest-path (APSP) variant of distance computation can be related to matrix exponentiation through the min-plus semiring, a recent line of works in distributed computing has explored the implication of this connection on approximation algorithms that utilize only $o(\text{poly}(n))$ Congested Clique rounds.

In this text, we analyze this line of works and discuss the future research directions of the field. The current state-of-the-art results exploit exact Congested Clique matrix multiplication to perform exact APSP in $O(n^{1/3})$ rounds, and $O(1)$-approximate APSP in $O(\log\log n)$ rounds or $O(\text{poly}\log n)$-approximate APSP in $O(1)$ rounds. It is of significant interest whether it is possible to perform $O(1)$-approximate APSP in $O(1)$ rounds as this would imply a drastic reduction in time complexity between exact APSP in $O(\text{poly}(n))$ rounds and approximate APSP (with a constant approximation factor) in constant rounds. In the rest of this text we highlight the results of this research direction and speculate with regards to future results.

Preliminaries

We begin by presenting the Congested Clique model, proceed to defining matrix multiplication (MM) and the all-pairs-shortest-path (APSP) problem in this setting, and conclude the preliminaries by showing some basic relationships between MM and APSP in Congested Clique.

In CONGEST [32], computational nodes and communication links between them are represented by a graph, G. Every node can perform unlimited computation, is initially only aware of its incident edges, and can communicate in synchronous rounds with its neighbors by sending each a $O(\log n)$-bit message per round. Typically, algorithms solve problems over G (e.g. distance computations), where the main complexity measure is the number of rounds an algorithm takes. The Congested Clique model [28] is similar to CONGEST, but separates the input from the communication topology: the input is a graph G, but the communication topology allows all nodes to communicate directly with one another.

Definition 1 (MM in Congested Clique)**.** *Let S, T be two n by n matrices such that, for all i, node i holds the i^{th} rows of S and T. Computing the product $P = S \cdot T$ in* Congested Clique *is achieved when, for all i, node i knows row i of P.*

Definition 2 (APSP in Congested Clique)**.** *Let G be an input graph. Computing the APSP over G is achieved when for for every nodes u, v, node u knows the distance from u to v in G.*

Brief Overview

Many recent Congested Clique papers study distance problems [7,10,18,21,24,29,31]. Specifically, [10, 21] exploit a well-known connection between distances and matrix multiplication – the n^{th} power of the adjacency matrix A of a graph, taken over the min-plus (or tropical) semiring, corresponds to shortest-path distances. Hence, iteratively squaring a matrix $\log n$ times gives the best known algorithms for APSP in Congested Clique, including (1) an $\widetilde{O}(n^{1/3})$ round algorithm for exact APSP in weighted directed graphs [10], (2) $O(n^{0.158})$ round algorithms for exact APSP in unweighted undirected graphs and $(1 + o(1))$-approximate APSP in weighted directed graphs [10], and (3) an $O(n^{0.2096})$ round algorithm for exact APSP in directed graphs with constant weights [21].

Faster approximations for larger constants are obtained by computing a k-spanner (sparse subgraph approximating distances by a factor of k), and having all nodes learn the spanner. Using the results of [31], this gives a $(2k - 1)$-approximation for APSP in $\widetilde{O}(n^{1/k})$ rounds, which is polynomial for any constant k. This raises the following fundamental question.

Question 1. *Are there constant-factor APSP approximations in sub-polynomial time?*

For SSSP, this is indeed possible [7, 23], with a gradient-descent-based algorithm obtaining a $(1 + \epsilon)$-approximation in $O(\epsilon^{-3}\text{polylog } n)$ rounds (even in BCC) [7].

In [27], we develop a line of sparsity aware matrix multiplication algorithms. The complexities of our algorithms do not depend on the sparsity structures of the matrices, rather, only on the total number of non-zero elements. We then apply our sparse matrix multiplication algorithms to efficiently implement basic primitives for distance computations. For instance, we compute for every node the distances to the $O(n^{2/3})$ nodes closest to it in $O(\text{poly} \log n)$ rounds. Together with other tools we develop, we show a $(2 + \epsilon)$ APSP approximation in $O(\log^2 n/\epsilon)$ rounds. This is the first sub-polynomial constant-factor APSP approximation, and is an *exponential* speedup over previous results. Following our work and using our distance tools, [16] show further improvement, bringing the round complexity down to $O(\text{poly} \log \log n)$, leading to the following question.

Question 2. *Is it possible to obtain an $o(\log \log n)$-round good APSP approximation?*

In [27], we enhance our tools using a technique we call *partition trees*, to develop a sparsity-aware sparsification tool to answer this in the affirmative. We construct $O(\log n)$-spanners and approximate APSP (with polylogarithmic approximation factors) in $O(1)$ rounds.

The Matrix Multiplication Cube

We describe the following visualization accompanying our techniques. How does one multiply matrices? By definition, given two n by n matrices, S and T, matrix P is their product if

$$\forall i, j \in [n] : P[i, j] = \sum_{k \in [n]} S[i, k]T[k, j] .$$

Typically, in introductory linear algebra courses, this is shown via a 2-dimensional depiction: "multiply (element-wise) a row of S by a column of T, and sum the values". Instead, we use the *3D Cube of Matrix Multiplication* (see e.g. [2, 3]). Formally, it is an $n \times n \times n$ cube, where entry (i, j, k) corresponds to $S[i][k]T[k][j]$. Two dimensions correspond to S and T, and each index in the third dimension represents a page, where P is the sum of all n pages.

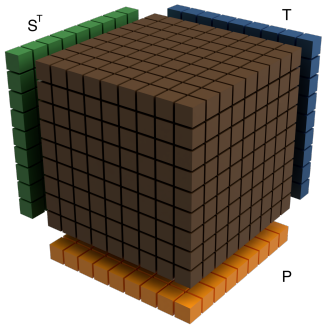


Figure 1: An illustration of the multiplication cube. The green (left) and blue (right) shapes correspond to S (transposed) and T, respectively, the brown (center) are point-wise multiplications, and the orange (bottom) are P. The value of every brown shape is the product of its green and blue projections. The value of every orange shape is the sum of the brown values above it.

The cube is useful for visualizing parallel and distributed algorithms. Assume k computational devices. A subcube $V_1 \times V_2 \times V_3$, where $V_1, V_2, V_3 \subseteq [n]$, corresponds to the task $S[V_1, V_2] \cdot T[V_2, V_3]$, where $S[V_1, V_2]$ is the submatrix limited to rows V_1 and columns V_2 of S, and similarly for T. Thus, a partition of n^3 into k subcubes breaks the larger matrix multiplication into smaller tasks.

Efficiently partitioning the cube is heavily dependant on the distributed setting considered. To compute the value at index (i, j, k), a device needs both its projections, $S[i][k]$ and $T[k][j]$. How can we factor in the (potentially different) sparsities of S and T? Is it easier to learn the projections onto S than those onto T? Computing the values of P requires computational devices computing intermediate values "above" one another (in the figure) to communicate – is this communication more expensive than that for learning the projections onto S and T?

We split our approaches for overcoming these challenges to input-sparsity awareness (S and T) and output-sparsity awareness (P). Designing output-sparsity aware algorithms is problematic as P, by definition, is not known in advance. Thus, this necessitates adaptive behaviour throughout the communication, before the output is fully computed. Our most complex algorithm, Filtered Sparse Matrix Multiplication, deals with dense output, where only some sparse set of it (matching specific predicates) is desired. Thus, already at the stage when only some of the intermediate computation is performed, we deduce which values of P we want to fully compute.

Note that in [27], we extend this further to deal with *subgraph existence problems* and not just matrix multiplication and distance computations. What if we use higher dimensions (4, 5, 6, etc.)? What if we reverse the information flow such that data flows *into* the cube from the

bottom (orange shapes) instead of *out*? What if, in some dimensions, we can perform *quantum* computation? For instance, reversing the information flow w.r.t. P turns a boolean matrix multiplication algorithm to one that finds all triangles (three fully connected nodes) in a graph. Thus, the cube amounts to *a unified perspective for seemingly unrelated distributed problems – intuition w.r.t. a property of the cube is carried over to many problems.*

Warmup: Using the MM Cube for Input Sparsity Awareness for exact APSP

In [11], we investigate how to perform matrix multiplication faster if it is given that the two input matrices are sparse. As a warmup, we leverage this to design faster exact APSP algorithms for sparse graphs. Later, we show how to extend the matrix multiplication algorithms to also take into account the sparsity of the output matrix, which allows approximating APSP exponentially faster, even on general graphs.

A central challenge in non-sequential matrix multiplication is high skew in input matrices, as Ballard et al. [5] describe in the parallel setting: "*[We] are not aware of any algorithms that dynamically determine and efficiently exploit the structure of general input matrices. In fact, a common technique of current library implementations is to randomly permute rows and columns of the input matrices in an attempt to destroy their structure and improve computational load balance.*" We show deterministic algorithms overcoming this, as well as other challenges which arise in distributed settings and not in parallel or sequential ones.

In Congested Clique, the round complexity is typically dominated by the node participating in the most communication. This leads us to defining two main goals: minimizing the total message count, and implementing load balancing mechanisms to ensure the round complexity is governed by the average number of messages each node communicates, and not the maximal.

On a high-level, our approach is threefold, with the first part minimizing the total number of messages sent, and the latter parts load balancing among the nodes.

First, split the $n \times n \times n$ matrix multiplication cube into n equally sized sub-cubes whose dimensions are determined dynamically, based on the sparsity of the input matrices. Fix some values a, b. We partition P into ab sub-matrices of size $n/a \times n/b$, denoted by $P_{i,j}$ for $i \in [a]$, $j \in [b]$, and assign n/ab nodes, denoted $N_{i,j}$, for computing $P_{i,j}$.

Second, notice that permutations of rows of S and columns of T result in a reversible permutation of P. Thus, we permute the S and T such that the number of non-zero entries required for computing each $n/a \times n/b$ sub-matrix is roughly the same for each sub-matrix. We call the two permuted matrices, S' and T', *sparsity-balanced matrices with respect to* (a, b). The rest of our algorithm deals with computing the product of such matrices.

Third, we assign the computation of pages of sub-matrices to nodes in a non-consecutive manner. Each $P_{i,j}$ is the sum of n sub-pages $P_{i,j,\ell}$. Each $v \in N_{i,j}$ computes some of the $P_{i,j,\ell}$ sub-pages and sums them locally. The local sums are then aggregated, to obtain $P_{i,j}$. We assign sub-pages to nodes in a non-consecutive manner, such that each node receives a roughly equal number of non-zero entries to compute its assigned sub-pages (See Figure 2).

While the above ensure nodes receive roughly the same number of messages, it is paramount that nodes also send a roughly equal number of non-zero matrix entries. We rearrange the entries of S and T held by each node such that each holds a roughly equal amount of non-zero entries. In this step, we do not permute S or T, rather, we merely redistribute their entries.

Crucially, these assignments are not global knowledge, leading to routing challenges. That is, for every $P_{i,j}$, nodes $N_{i,j}$ decide which matrix entries are received by which node, yet, this is

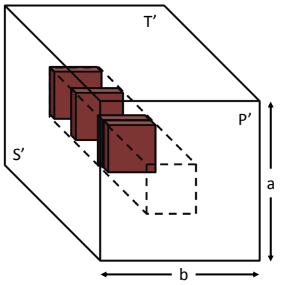


Figure 2: An illustration of the multiplication cube for $P' = S'T'$. Each sub-matrix is assigned to n/ab nodes, with a not necessarily consecutive page assignment that is computed on-the-fly to minimize communication.

unknown to other nodes who need to send the entries. Likewise, the redistribution of S and T is not known to all. However, clearly, a node must know the destination of each message it sends. We design our solution such that every node computes a small set of nodes potentially holding information it requires, and request it from them. Upon receipt of the request, the nodes can compute whether they actually have the relevant information, and, if so, send it over.

For matrix M, let $nz(M)$ denote the total number of non-zero entries in M, and let $\rho_M = \lceil nz(M)/n \rceil$ denote the density of M. Our first main contribution is the following.

Theorem 1. *Given two $n \times n$ matrices S and T, it is possible to deterministically compute $P = ST$ over a semiring, within $O((\rho_S \rho_T)^{1/3}/n^{1/3} + 1)$ rounds in* Congested Clique.

An important case of Theorem 1, especially when squaring the adjacency matrix of a graph, is when the sparsities of the input matrices are roughly the same.

Corollary 2. *Given two $n \times n$ matrices S and T, where $O(nz(S)) = O(nz(T)) = m$, it is possible to compute $P = ST$ over a semiring, in $O(m^{2/3}/n + 1)$ rounds in* Congested Clique.

For $m = O(n^2)$, Corollary 2 gives the same $O(n^{1/3})$ rounds complexity as that of [10], the state-of-the-art for non-sparse matrix multiplication. Our algorithm is fast also when only one of the matrices is sparse, as stated in the following.

Corollary 3. *Given two $n \times n$ matrices S and T, where $\min\{nz(S), nz(T)\} = m$, it is possible to compute $P = ST$ over a semiring, in $O((m/n)^{1/3} + 1)$ rounds in* Congested Clique.

This allows computing powers that are larger than 2 of a sparse input matrix. We cannot repeatedly square a matrix, as this may require multiplying dense matrices, yet, we can repeatedly increase its power by 1. This gives the following for exact APSP, whose comparison to the state-of-the-art depends on the trade-off between the number of edges and the graph diameter.

Theorem 4. *Given an unweighted graph G, there is a deterministic algorithm that computes APSP in $O(D((m/n)^{1/3} + 1))$ rounds in the* Congested Clique *model.*

To compare, the best known complexity for general graphs is $O(n^{1-2/\omega})$ [10] (currently roughly $O(n^{0.158})$). For a graph with $m = o(n^{4-6/\omega}/D^3)$ (currently $o(n^{1.474}/D^3)$), our algorithm is faster.

$O(1)$ APSP approximation in $O(\text{poly}\log\log n)$ rounds

In [9], we turn our attention to general graphs, and aim to compute a constant approximation of APSP in sub-polynomial rounds.

Distance Products. We start from the basic idea of using matrix multiplication to compute distances. Specifically, if A is the weighted adjacency matrix of a graph G, it is well known that distances in G can be computed by iterating the distance product $A \star A$, defined as

$$(A \star A)[i,j] = \min_k (A[i,k] + A[k,j]),$$

that is, the matrix multiplication over the min-plus semiring.

However, as $A \star A$ can be dense even if A is sparse (e.g. a star graph), iterative squaring is not guaranteed to be efficient. Moreover, our goal is to compute distances in general graphs and so we do not even assume A itself is sparse. We thus take a step back, first showing several distance computation building blocks, before directly tackling end-problems such as APSP.

Our Distance Tools. The key observation is that building blocks for distance computation are actually based on computations in sparse graphs or subgraphs. Concrete examples include:

- k-NEAREST: Compute distances for each node to the k nodes closest to it.
- (S,d,k)-SOURCE DETECTION: Given a set of sources S, compute the distances for each node to the k sources closest to it, using paths of at most d hops.
- DISTANCE THROUGH SETS: Given a set of nodes S and distances to all nodes in S, compute the distances between all nodes using paths through nodes in S.

For all of these problems, there is a degree of sparsity we can hope to exploit if k or $|S|$ are small. For example, the (S,d,k)-SOURCE DETECTION problem, requires the multiplication of a dense adjacency matrix with a possibly sparse matrix, depending on the size of S. However, for any S of polynomial size, the round complexity of the input sparsity aware algorithm is polynomial. An interesting property in this problem is that the output matrix is also sparse, giving room for improvement. As another example, in k-NEAREST both input matrices are sparse, which makes it fast using the previous sparse matrix multiplication algorithm. However, this does not exploit the sparsity of this problem to the end: we are interested only in computing the k nearest nodes to each node, hence there is no need to compute the full output matrix. The challenge in this case is that we do not know the identity of the k closest nodes before the computation.

To exploit this sparsity we design new matrix multiplication algorithms that, in particular, can sparsify the output matrix throughout its computation, and get a complexity that depends only on the size of the output we are interested in. The core reason that significant challenges arise in output sparsity awareness w.r.t. input sparsity awareness is that the input matrices are known beforehand, while clearly this does not hold in the output case. Thus, we are required to recognize patterns in the output during the computation and perform actions affected by these patterns. In particular, our approaches uses both *binary search and sorting*, in a novel way, to efficiently perform the summation step of the matrix multiplication, while abstracting away effects of the output patterns. We obtain the following matrix multiplication variants.

- One variant assumes that the sparsity of the output matrix is known.

- The other sparsifies the output on the fly, keeping the ρ_P smallest entries in each row.

For these two scenarios, we obtain round complexities

$$O\Big(\frac{(\rho_S\rho_T\rho_P)^{1/3}}{n^{2/3}}+1\Big), \qquad \text{and} \qquad O\Big(\frac{(\rho_S\rho_T\rho_P)^{1/3}}{n^{2/3}}+\log n\Big),$$

respectively, improving over our input sparsity aware matrix multiplication for $\rho_P = o(n)$.

Applications of Sparse Matrix Multiplication. Using output sensitive sparse matrix multiplication, we obtain faster distance tools:

- We solve k-NEAREST in $O\left(\left(\frac{k}{n^{2/3}}+\log n\right)\log k\right)$ rounds.
- We solve (S,d,k)-SOURCE DETECTION in $O\left(\left(\frac{m^{1/3}|S|^{2/3}}{n}+1\right)d\right)$ rounds, where m is the number of edges in the output graph; note that dependence on d becomes linear in order to exploit the sparsity.

In concrete terms, with these output sensitive distance tools we still get subpolynomial running times even when the parameters are polynomial. For example, we get the distances to the $\widetilde{O}(n^{2/3})$ closest nodes in $\widetilde{O}(1)$ rounds. Note that though our final results are only for undirected graphs, these distance tools work for directed, weighted graphs.

Hopsets. An issue with our (S,d,k)-SOURCE DETECTION algorithm is that in order to exploit the sparsity of the matrices, we must perform d multiplications to learn the distances of nodes at hop-distance at most d from S. Hence, to learn the distances of all nodes from S, we need to do n multiplications, which is no longer efficient. To overcome this challenge, we use hopsets. Given a (β,ϵ)-hopset H, it is enough to look only at β-hop distances in $G \cup H$ to approximate distances by a factor of $(1+\epsilon)$. Using our source detection algorithm together with a hopset allows getting an efficient algorithm for approximating distances, as long as β is small enough.

However, the round complexities of all current hopset constructions [17, 18, 23] is at least $O(\rho)$ for a hopset of size $n\rho$. This is a major obstacle for efficient shortest paths algorithms, since, based on recent existential results, there are no hopsets where both β and ρ are polylogarithmic [1]. Nevertheless, we show that our new distance tools build a hopset in a time that does not depend on its size. In particular, we show how to implement a variant of the recent hopset construction of in [18] in $O(\frac{\log^2 n}{\epsilon})$ rounds. The size of our hopset is $\widetilde{O}(n^{3/2})$, hence constructing it using previous algorithms requires at least $\widetilde{O}(\sqrt{n})$ rounds.

Applying the Distance Tools. As a direct application of our source detection and hopset algorithms, we obtain an algorithm for computing distances from k sources at once (k-SSP). This is the first sub-polynomial result, with such approximations, for polynomial k.

Theorem 5. *There is a deterministic* $(1+\epsilon)$*-approximation algorithm for weighted undirected* k-SSP *that takes*

$$O\Big(\Big(\frac{k^{2/3}}{n^{1/3}}+\log n\Big)\cdot\frac{\log n}{\epsilon}\Big)$$

rounds in the Congested Clique*, where* S *is the set of sources. In particular, the complexity is* $O(\frac{\log^2 n}{\epsilon})$ *as long as* $k = O(\sqrt{n}\cdot(\log n)^{3/2})$.

In turn, this forms the basis of our $(3+\epsilon)$-approximation for weighted APSP. To obtain a $(2+\epsilon)$-approximation for unweighted APSP, the idea is to deal separately with paths containing a high-degree node and paths without. A crucial ingredient is showing that in sparser graphs we can efficiently compute distances to a larger set S.

Theorem 6. *There is a deterministic $(2+\epsilon)$-approximation algorithm for unweighted undirected APSP in the* Congested Clique *model that takes $O(\frac{\log^2 n}{\epsilon})$ rounds.*

Our approximation is almost tight for sub-polynomial algorithms in the following sense. As noted by [26], a $(2-\epsilon)$-approximate APSP in unweighted undirected graphs is essentially equivalent to fast matrix multiplication, so obtaining a better approximation in complexity below $O(n^{0.158})$ would result in a faster algorithm for non-sparse matrix multiplication in the Congested Clique. Likewise, a sub-polynomial-time algorithm with *any* approximation ratio for directed graphs would give a faster matrix multiplication algorithm [14].

As stated above in the overview, this concludes the first sub-polynomial constant-factor APSP approximation, showing an *exponential* speedup over previous results which all required polynomial rounds for such an approximation. Following this work and using the above described distance tools, [16] bring the complexity down to $O(\text{poly}\log\log n)$. In weighted graphs, an $O(\log^{1+\epsilon} n)$-approximation to APSP is known in a similar round complexity [8].

$O(\text{poly}\log n)$ APSP approximation in $O(1)$ rounds

In [15], we ask whether it is possible to spend *just* $O(1)$ rounds in order show a good approximation of APSP. We answer this by showing poly-logarithmic approximations to APSP in such a round complexity.

Spanners. A k-spanner is a sparse subgraph, preserving distances up to a factor of k. Any graph has a $(2k-1)$-spanner with $O(n^{1+1/k})$ edges, which is assumed to be tight by the Erdős Girth Conjecture [19]. We aim for spanners with $O(n)$ edges, requiring $k=\Theta(\log n)$. The classic algorithm of [6] builds $(2k-1)$-spanners with $O(kn^{1+1/k})$ edges in expectation, and is simulated in $O(k)$ rounds. Faster, poly$(\log k)$-round algorithms appear in [8, 31]. Specifically, [31] shows a randomized (deterministic) construction of $(2k-1)$-spanners ($O(k)$-spanners) with $\widetilde{O}(n^{1+1/k})$ edges ($O(kn^{1+1/k})$ edges). For weighted undirected graphs, [8] show a randomized construction of $O(k^{1+o(1)})$-spanners with $O(n^{1+1/k}\cdot\log k)$ edges.

The Locality Barrier. Intuitively, these take poly$(\log k)$ rounds as the locality of spanners is linear in k. In the LOCAL model, where a node can learn its entire t-neighborhood in t rounds, constructing $(2k-1)$-spanners with $O(n^{1+1/k})$ edges takes $\Omega(k)$, assuming the Erdős Girth Conjecture [13]. The connection between LOCAL and Congested Clique is captured via graph exponentiation, whereby nodes learn their 2^i-neighborhoods in round i. This requires much bandwidth, nevertheless, is sometimes used with other ideas, leading to $O(\log t)$ algorithms in Congested Clique based on t-round algorithms in LOCAL (e.g. [16, 31]). In variants of MPC, this approach is conditionally tight [22], for certain algorithms.

We break through the locality barrier, by utilizing techniques from our distances computations (above), our partition trees tool (see **??**), as well as recent developments for computing a minimum spanning tree (MST) in Congested Clique in $O(1)$ rounds [25, 30].

A powerful ingredient is our sparsification tool, Theorem 7, based on our partition trees (**??**), that finds spanners for graphs F connecting some nodes in G.

Theorem 7. *Let $G = (V, E)$ be a* Congested Clique*, and $F = (V_F, E_F)$ a graph with $V_F \subseteq V$, $|V_F| = N$ nodes, $|E_F| = M$ edges, and maximum degree Δ_F. There is an $O(\frac{M^{1/3} \cdot N^{2/3}}{n} + 1)$-round algorithm in G that finds a $(2k-1)$-spanner for F with $O(M^{1/3} \cdot N^{2/3+1/k})$ edges.*

In a recent work on MST computation, given $F = (V_F, E_F)$, $|V_F| = N$, $|E_F| = M$, [30] uses the following. Let $d = 2M/N$ and partition V_F into $S_1, \cdots, S_{\sqrt{d}}$, where $|S_i| = O(N/\sqrt{d})$ and $|E(S_i, S_j)| = O(N)$. Then, the edges $E(S_i, S_j)$ are sent to some node, which replaces them with an MST, with $|S_i| + |S_j|$ edges, on the graph induced by $S_i \times S_j$. Since $|S_i \cup S_j| = O(N/\sqrt{d})$, each MST has $O(N/\sqrt{d})$ edges, and all MSTs have $d \cdot O(N/\sqrt{d}) = O(\sqrt{M \cdot N})$ edges in total.

We show Theorem 7 by partitioning into $d^{1/3}$ sets, and follow the notion where each node sparsifies the edges it gets, by returning a spanner (instead of an MST). Our partitioning in [15] is randomized, and using our partition trees technique (see **??**), we show a deterministic version. The key behind this is that applying our partition trees technnique on the "subgraph" $H = (V_H, E_H)$ which is simply one edge (i.e. $V_H = \{a, b\}, E_H = \{\{a, b\}\}$) redistributes the edges of $G = (V, E)$ such that every v knows some set of edges E_v, where $\{E_v | v \in V\}$ is a partition of E, and the amount of nodes incident to each E_v is rather small. Based on this approach, we show a very powerful partitioning which leads to Theorem 7.

Unweighted Graphs We apply Theorem 7 on cluster graphs generated by a smart sampling procedure. Let $d = 2m/n$. We construct a cluster graph C. Find a hitting set $D \subseteq V$ (every node is either in D or has at least one neighbor in D) of size $O(n \log d/d)$. Then, each node in D is denoted the center of a cluster in C, and each node in $V \setminus D$ joins the cluster of one of its neighbors in D. For any two clusters C_1 and C_2 in C, connect them with an edge if, in G, any node in C_1 is connected to any node in C_2. As C has $M \leq m$ edges but only $N = O(n \log d/d)$ nodes, we can apply Theorem 7 on C, to get a spanner with $\ell = O(M^{1/3} \cdot N^{2/3+1/k}) = O((dn)^{1/3} \cdot (n \log d/d)^{2/3+1/k}) = O(n^{1+1/k})$ edges in $O(\frac{M^{1/3} \cdot N^{2/3}}{n}) = O(1)$ rounds.

Any α-spanner H_C of C with ℓ edges can be translated to an $O(\alpha)$-spanner H for G with $\ell + n$ edges. Replace each edge in H_C by an edge in G that connects two nodes in the corresponding clusters. Then, for each $v \in V$, add to H the edge that connects it to the center of the cluster v belongs to. H is an $O(\alpha)$-spanner for G, as any path P_C of length α in H_C translates to a path P in H through $\alpha + 1$ clusters. As the radius of each cluster is 1, P has length $O(\alpha)$.

However, it is hard to find a hitting set with $|D| = O(n \log d/d)$. In graphs with minimum degree d, a sampling procedure suffices, yet d is our average degree. Thus, we bucket G, where bucket i contains edges E_i, incident to nodes with degree in $[2^i, 2^{i+1})$, and run the above, for bucket i[1] computing C_i and H_{C_i}. The size of all spanners, where Δ is the maximum degree, is:

$$|\bigcup_{i=0}^{\log \Delta} H_i| = |H_0| + \sum_{i=1}^{\log \Delta} \mathrm{O}((ni/2^i)^{2/3+1/k}(n2^i)^{1/3}) \leq \mathrm{O}(n) + \sum_{i=1}^{\log \Delta} \mathrm{O}(n^{1+1/k}) i^{2/3}/2^{i/3} = \mathrm{O}(n^{1+1/k})$$

Finally, we convert H_{C_i} to H_i which is a spanner for edges E_i of G, and $H = \bigcup_i H_i$ is a spanner of G, as required. However, as above, $|H_i| \leq |H_{C_i}| + n$, where the $+n$ is due to connecting each node its cluster center. This may result in $|H| = O(n^{1+1/k} + n \log \Delta)$, instead of $O(n^{1+1/k})$. To solve this, we construct the graphs C_i such that for each $v \in V$, all clusters v belongs to have the same center. Hence, each node adds at most only one edge in total, which means that across all H_i at most n edges are added, and not n edges per H_i.

[1] Except for $i = 0$, where we define H_{C_i} as all edges incident to a node of degree at most 2.

Theorem 8. *Given k and an undirected unweighted graph G, there is an $O(1)$-round* Congested Clique *algorithm constructing an $O(k)$-spanner for G with $O(n^{1+1/k})$ edges, w.h.p.*

Choosing $k = \Theta(\log n)$ gives an $O(\log n)$-spanner of size $O(n)$, thus implying the following.

Theorem 9. *Given an undirected unweighted graph G, there is an $O(1)$-round* Congested Clique *algorithm computing an $O(\log n)$-approximation of APSP w.h.p.*

Weighted Graphs We develop two additional tools to extend our results to weighted graphs. First, we split the edges into buckets of exponentially increasing weights $B_l = \{e \mid 2^l \leq w(e) < 2^{l+1}\}$, and construct, in parallel, a spanner for each bucket using the unweighted algorithm. The union of these spanners is an $O(k)$-spanner for G, yet with a total of $O(n^{1+1/k} \log n)$ edges. To obtain a spanner with $O(n^{1+1/k})$ edges (yet $O(k \log n)$ stretch), we draw a connection to MSTs. We construct an MST using [30] and use it to contract the graph to $n/\log n$ nodes, while preserving distances up to a factor of $O(\log n)$. This is done by replacing low diameter subtrees of the MST, each with a single node, and due to the property that the edges of the MST cannot be the heaviest along a cycle, we show that graph distances are stretched only by $O(\log n)$. Then, we execute the above spanner algorithm on the contracted graph, and as the number of nodes is $n/\log n$, the resulting spanner has $O((n/\log n)^{1+1/k} \log (n/\log n)) = O(n^{1+1/k})$ edges.

Theorem 10. *Given an undirected weighted graph G, there is an $O(1)$-round algorithm in the* Congested Clique *model that computes an $O(\log^2 n)$-approximation of APSP, w.h.p.*

Conclusion

The research community has recently made rapid progress in APSP approximations in the Congested Clique model, mainly based on algorithms solving variants of matrix multiplication. Currently, the state-of-the-art is a constant approximation in $O(\text{poly} \log \log n)$ rounds, or an $O(\log n)$ approximation in constant rounds ($O(\text{poly} \log n)$ if the graph is weighted). We hypothesize that the community is close to achieving a constant approximation in constant rounds barring a few additional insights. This is certain to lead to several new exciting research directions in the upcoming years.

Concretely, it is interesting to see whether the techniques behind the two extreme results detailed above (constant approximation in $O(\text{poly} \log \log n)$ rounds or $O(\text{poly} \log n)$ approximation in constant rounds) can be merged in order to show a better overall result. These are active research directions which the community is currently investigating.

References

[1] Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. *SIAM Journal on Computing*, 47(6):2203–2236, 2018.

[2] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development*, 39(5):575–582, 1995. doi:10.1147/rd.395.0575.

[3] Alok Aggarwal, Ashok K. Chandra, and Marc Snir. Communication complexity of prams. *Theoretical Computer Science*, 71(1):3–28, 1990. doi:https://doi.org/10.1016/0304-3975(90)90188-N.

[4] Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539, 2021. doi:10.1137/1.9781611976465.32.

[5] Grey Ballard, Aydin Buluç, James Demmel, Laura Grigori, Benjamin Lipshitz, Oded Schwartz, and Sivan Toledo. Communication optimal parallel multiplication of sparse random matrices. In *Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures, (SPAA)*, pages 222–231, 2013. URL: http://doi.acm.org/10.1145/2486159.2486196, doi:10.1145/2486159.2486196.

[6] Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007.

[7] Ruben Becker, Sebastian Forster, Andreas Karrenbauer, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. *SIAM Journal on Computing*, 50(3):815–856, 2021. doi:10.1137/19M1286955.

[8] Amartya Shankha Biswas, Michal Dory, Mohsen Ghaffari, Slobodan Mitrovic, and Yasamin Nazari. Massively parallel algorithms for distance approximation and spanners. *SPAA 2021*, 2021.

[9] Keren Censor-Hillel, Michal Dory, Janne H. Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. *Distributed Computing*, 2020. doi:10.1007/s00446-020-00380-5.

[10] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. *Distributed Computing*, 32(6):461–478, 2019. doi:10.1007/s00446-016-0270-2.

[11] Keren Censor-Hillel, Dean Leitersdorf, and Elia Turner. Sparse matrix multiplication and triangle listing in the congested clique model. *Theoretical Computer Science*, 809:45–60, 2020. doi:10.1016/j.tcs.2019.11.006.

[12] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990. doi:10.1016/S0747-7171(08)80013-2.

[13] Bilel Derbel, Cyril Gavoille, David Peleg, and Laurent Viennot. On the locality of distributed sparse spanner construction. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing (PODC)*, pages 273–282, 2008.

[14] Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000.

[15] Michal Dory, Orr Fischer, Seri Khoury, and Dean Leitersdorf. Constant-round spanners and shortest paths in congested clique and mpc. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, page 223–233, 2021. doi:10.1145/3465084.3467928.

[16] Michal Dory and Merav Parter. Exponentially faster shortest paths in the congested clique. In *PODC '20: ACM Symposium on Principles of Distributed Computing, August 3-7, 2020*, pages 59–68. ACM, 2020. doi:10.1145/3382734.3405711.

[17] Michael Elkin and Ofer Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. *SIAM J. Comput.*, 48(4):1436–1480, 2019. doi:10.1137/18M1166791.

[18] Michael Elkin and Ofer Neiman. Linear-size hopsets with small hopbound, and constant-hopbound hopsets in RNC. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 333–341, 2019. doi:10.1145/3323165.3323177.

[19] Paul Erdős. Extremal problems in graph theory. In *Theory Of Graphs And Its Applications, Proceedings of Symposium Smolenice*, pages 29–36. Publ. House Cszechoslovak Acad. Sci., Prague, 1964.

[20] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. ISSAC 2014*, pages 296–303, 2014. doi:10.1145/2608628.2608664.

[21] François Le Gall. Further algebraic algorithms in the congested clique model and applications to graph-theoretic problems. In *Proceedings of the 30th International Symposium on Distributed Computing (DISC)*, pages 57–70, 2016. doi:10.1007/978-3-662-53426-7_5.

[22] Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional hardness results for massively parallel computation from distributed lower bounds. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1650–1663. IEEE, 2019.

[23] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A deterministic almost-tight distributed algorithm for approximating single-source shortest paths. *SIAM J. Comput.*, 50(3), 2021. doi:10.1137/16M1097808.

[24] Stephan Holzer and Nathan Pinsker. Approximation of distances and shortest paths in the broadcast congest clique. In *Proc. OPODIS 2015*, 2015. doi:10.4230/LIPIcs.OPODIS.2015.6.

[25] Janne H. Korhonen. Deterministic MST sparsification in the congested clique. *CoRR*, abs/1605.02022, 2016.

[26] Janne H. Korhonen and Jukka Suomela. Towards a complexity theory for the congested clique. In *Proc. SPAA 2018*, pages 163–172, 2018. doi:10.1145/3210377.3210391.

[27] Dean Leitersdorf. *Fast Distributed Algorithms via Sparsity Awareness*. PhD thesis, Technion - Israel Institute of Technology, Israel, 2022. URL: https://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi/2022/PHD/PHD-2022-09.

[28] Zvi Lotker, Boaz Patt-Shamir, Elan Pavlov, and David Peleg. Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. *SIAM J. Comput.*, 35(1):120–131, 2005. doi:10.1137/S0097539704441848.

[29] Danupon Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *Proc. STOC 2014*, pages 565–573. ACM, 2014.

[30] Krzysztof Nowicki. A deterministic algorithm for the MST problem in constant rounds of congested clique. *STOC 2021*, 2021.

[31] Merav Parter and Eylon Yogev. Congested clique algorithms for graph spanners. In *Proc. DISC 2018*, pages 40:1–40:18, 2018. doi:10.4230/LIPIcs.DISC.2018.40.

[32] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, USA, 2000.

[33] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969. doi:10.1007/BF02165411.

[34] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proceedings of the 44th ACM Symposium on Theory of Computing (STOC)*, pages 887–898, 2012. URL: http://doi.acm.org/10.1145/2213977.2214056, doi:10.1145/2213977.2214056.

THE LOGIC IN COMPUTER SCIENCE COLUMN

BY

YURI GUREVICH

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

BEATCS no 141

WHAT ARE KETS?

Yuri Gurevich
Computer Science & Engineering
University of Michigan

Andreas Blass
Mathematics
University of Michigan

Abstract

According to Dirac's bra-ket notation, in an inner-product space, the inner product $\langle x \,|\, y\rangle$ of vectors x, y can be viewed as an application of the bra $\langle x|$ to the ket $|y\rangle$. Here $\langle x|$ is the linear functional $|y\rangle \mapsto \langle x \,|\, y\rangle$ and $|y\rangle$ is the vector y. But often — though not always — there are advantages in seeing $|y\rangle$ as the function $a \mapsto a \cdot y$ where a ranges over the scalars. For example, the outer product $|y\rangle\langle x|$ becomes simply the composition $|y\rangle \circ \langle x|$. It would be most convenient to view kets sometimes as vectors and sometimes as functions, depending on the context. This turns out to be possible.

While the bra-ket notation arose in quantum mechanics, this note presupposes no familiarity with quantum mechanics.

1 The question

Q[1]: Gentlemen, I have a question for you. But first I need to motivate it and explain where I am coming from.

The question is related to the so-called inner-product spaces which are vector spaces over the field $\mathbb{R}$ of real numbers or the field $\mathbb{C}$ of complex numbers, furnished with inner product $\langle x \,|\, y\rangle$, also known as scalar product. Euclidean spaces and (more generally) Hilbert spaces are the most familiar examples of inner-product spaces. I'll stick to the case of $\mathbb{C}$ which is of greater interest to me.

Let $\mathcal{H}$ be an inner-product space over $\mathbb{C}$, and let x, y range over the vectors in $\mathcal{H}$. A vector x gives rise to the linear functional $y \mapsto \langle x \,|\, y\rangle$ that maps any vector y to the scalar $\langle x \,|\, y\rangle$. This linear functional is called a *bra* and denoted $\langle x|$ in Dirac's bra-ket notation, introduced by Paul Dirac [1]. The vector y is called a *ket* and denoted $|y\rangle$. Thus the inner product $\langle x \,|\, y\rangle$ can be viewed as the application $\langle x| \, |y\rangle = \langle x|(|y\rangle)$ of the bra $\langle x|$ to the ket $|y\rangle$.

[1] Quisani is a former student of the first author.

By the way a side question occurs to me. If kets are vectors and bras are linear functionals, then $\langle x \,|\, y\rangle$ can just be *defined* as the application $\langle x|\,|y\rangle$ in any vector space V, rather than presumed to exist. Does the inner product contribute anything?

A[2]: It does. It provides a particular embedding $|x\rangle \mapsto \langle x|$ of our vector space $\mathcal{H}$ to the vector space of linear functionals on $\mathcal{H}$, which is the dual of $\mathcal{H}$ in the theory of vector spaces. Notice that the axiom $\langle x \,|\, x\rangle \geqslant 0$ makes good sense for inner product spaces but not if we have no connection between $|x\rangle$ and $\langle x|$. If $\mathcal{H}$ is a Hilbert space, in particular if $\mathcal{H}$ is finite dimensional, this embedding is an isomorphism.

Q: Thanks. Let me proceed to my main question. As I said, kets are vectors according to Dirac, and the point of view that kets are vectors is ubiquitous. Here is a quote from my favorite textbook on quantum computing: "The notation $|\cdot\rangle$ is used to indicate that the object is a vector" [2, p. 62].

But recently I watched a recorded lecture [3] by Reinhard Werner, a professor of physics, who defined $|y\rangle$ as the linear function $a \mapsto a \cdot y$ from scalars to vectors, so that y is $|y\rangle(1)$.

A: Did Prof. Werner compare the definitions?

Q: No, there was only one definition of kets in his lecture. But I think that the outer product $|y\rangle\langle x|$ demonstrates an advantage of his approach. In the traditional approach, $|y\rangle\langle x|$ is defined to be a (linear) function by fiat. For example, Nielsen and Chuang [2, p. 68] write:

> "Suppose $|v\rangle$ is a vector in an inner product space V, and $|w\rangle$ is a vector in an inner product space W. Define $|w\rangle\langle v|$ to be the linear operator from V to W whose action is defined by $\big(|w\rangle\langle v|\big)(|v'\rangle) \equiv |w\rangle \langle v \,|\, v'\rangle = \langle v \,|\, v'\rangle\, |w\rangle$."

I do not know how consistent Prof. Werner is in using his definition of kets, but for the purpose of this discussion let me introduce a purely functional approach where a ket $|y\rangle$ is always the function $a \mapsto a \cdot y$. In this approach, $|y\rangle\langle x|$ is simply the composition $|y\rangle \circ \langle x|$ of two linear functions and thus $|y\rangle\langle x|$ is naturally a linear function, the same function that Nielsen and Chuang define.

At last, I come to my question: Is a ket a vector or a function? It cannot be both, can it?

A: Well, abuse of notation is common in mathematics, especially if the meaning is obvious from the context.

Q: Mathematically, the purely functional approach is attractive. Since composition of

[2]The authors speaking one at a time

functions is associative, we can drop the parentheses in expressions like

$$|u\rangle\langle v|\,|w\rangle\langle x|\,|y\rangle\langle z|.$$

But the purely functional approach has some problems. Consider the scalar product $\langle x\,|\,y\rangle$ for example. The composition $\langle x| \circ |y\rangle$ is a linear operator on $\mathbb{C}$, not a scalar. We can define, by fiat, that the original $\langle x\,|\,y\rangle$ is, in the purely functional approach, $(\langle x| \circ |y\rangle)\,(1)$. It would be more natural, of course, to view $|y\rangle$ as a vector in the context of scalar product.

I wonder whether one can take advantage of both approaches even if this leads to abuse of notation. Maybe there is a resolution of this abuse so that the intended meaning is obvious from the context.

2 Dirac terms

A: It seems that there are such resolutions. We need some analysis to understand what is going on. This discussion may be more pedantic than usual.

To keep the notation simple, we restrict attention to kets and bras over the same inner-product space $\mathcal{H}$. For the purpose of the analysis, we put forward the following tentative convention.

Tentative Convention. Every occurrence of a ket is marked as a *vector ket* or a *function ket*. If y is a vector then the vector ket $|y\rangle$ denotes the vector y but the corresponding function ket $|y\rangle$ denotes the function $a \mapsto a \cdot y$. More generally, for any label L, the vector ket $|L\rangle$ denotes some vector $\vec{v}$ in $\mathcal{H}$, while the function ket $|L\rangle$ denotes the function $a \mapsto a \cdot \vec{v}$, from $\mathbb{C}$ to $\mathcal{H}$, for the same vector $\vec{v}$. ◁

Q: You can avoid marking by declaring that, by default, kets are function kets; the corresponding vector ket is $|L\rangle(1)$.

A: This is correct. We stick to marking because of its symmetry. Your proposal may give an impression of bias in favor of function kets.

You have mentioned inner products $\langle x\,|\,y\rangle$ and outer products $|y\rangle\langle x|$. Let's consider more general products of alternating kets and bras.

Dirac terms: syntax. Kets and bras are *Dirac characters*. By a *Dirac term*[3] we mean a nonempty sequence of Dirac characters where kets and bras alternate; the sequence is

often furnished with parentheses.

More formally, Dirac terms are defined inductively. Dirac characters are terms. A concatenation $s_1 s_2$ of Dirac terms s_1, s_2 where kets and bras alternate is a Dirac term.

By default, a Dirac term s comes with enough parentheses to parse s, i.e. to determine how s is constructed from kets and bras by means of concatenation.

Q: Suppose some ket $|L\rangle$ occurs more than once in a Dirac term. Can some occurrences of $|L\rangle$ be vector kets and some function kets?

A: Sure, why not?

Dirac terms: semantics.

Q: Since kets are disambiguated in the Tentative Convention, semantics seems obvious.

A: It is obvious, but we need to spell out details in order to pursue our analysis.

By induction, we define the intended values $\mathrm{Val}(s)$ of (or denoted by) Dirac terms s and check that the the following equivalences hold.

E1 $\mathrm{Val}(s)$ is a linear function with domain $\mathcal{H} \iff s$ ends with a bra, and
$\mathrm{Val}(s)$ is a linear function with domain $\mathbb{C} \iff s$ ends with a function ket.
E2 $\mathrm{Val}(s)$ is a vector or a scalar $\iff s$ ends with a vector ket.
E3 $\mathrm{Val}(s)$ is a scalar or a scalar-valued function $\iff s$ starts with a bra.
E4 $\mathrm{Val}(s)$ is a vector or a vector-valued function $\iff s$ starts with a ket.

The value $\mathrm{Val}\big(\langle x|\big)$ of a bra $\langle x|$ is the linear $\mathcal{H} \to \mathbb{C}$ function denoted by $\langle x|$. The ket values are described in the Tentative Convention above. The equivalences E1–E4 are obvious in these cases.

Let s be a concatenation $s_1 s_2$ of constituent subterms (which satisfy E1–E4 of course), and let V_1, V_2 be the values of s_1, s_2 respectively. Four cases arise depending on whether V_1, V_2 are functions or not.

FF If V_1 is a function and V_2 is a function, then $\mathrm{Val}(s)$ is the composition $V_1 \circ V_2$, so that $\mathrm{Val}(s)$ is a function whose domain is that of V_2 and

$$\mathrm{Val}(s)(A) = \big(V_1 \circ V_2\big)(A) = V_1(V_2(A)) \qquad \text{for all } A \in \mathrm{Dom}(V_2).$$

[3]While logicians speak about terms, computer scientists speak about expressions. Here we use the logicians' vocabulary for a very utilitarian reason: "term" is shorter than "expression."

FN If V_1 is a function but V_2 is not then

$$\mathrm{Val}(s) = V_1(V_2).$$

NF If V_1 isn't a function but V_2 is then $\mathrm{Val}(s)$ is the function $V_1 \cdot V_2$ so that $\mathrm{Val}(s)$ is a function whose domain is that of V_2 and

$$\mathrm{Val}(s)(A) = \big(V_1 \cdot V_2\big)(A) = V_1 \cdot (V_2(A)) \qquad \text{for all } A \in \mathrm{Dom}(V_2).$$

NN If neither V_1 nor V_2 is a function then

$$\mathrm{Val}(s) = V_1 \cdot V_2.$$

Q: I see that you overload the multiplication symbol $\cdot$ with different types.

A: We do. But notice that at least one of the factors is always a scalar. It is very common to multiply scalars, vectors, and linear functions by scalars. But let's check that our definitions make sense and that the equivalences E1–E4 hold.

Clauses FF and FN make sense in that V_2 belongs to or takes values in $\mathrm{Dom}(V_1)$. Indeed, by E1, s_1 ends with a bra or a function ket. If s_1 ends with a bra, then $\mathrm{Dom}(s_1) = \mathcal{H}$ by E1 for s_1, and s_2 starts with a ket, and the desired property follows from E4 for s_2. If s_1 ends with a function ket, then $\mathrm{Dom}(s_1) = \mathbb{C}$ by E1 for s_1, and s_2 starts with a bra, and the desired property follows from E3 for s_2.

Clauses NF and NN also make sense, which is obvious if V_1 is a scalar. Otherwise V_1 is a vector and it suffices to check that V_2 is a scalar or scalar-valued function. By E2, s_1 ends with a ket. Hence s_2 starts with a bra. Use E3 for s_2.

It remains to check that $\mathrm{Val}(s)$ satisfies the equivalences E1–E4. By FF–NN, $\mathrm{Val}(s)$ is a function if and only if V_2 is a function, and if $\mathrm{Val}(s)$ is a function then its domain is $\mathrm{Dom}(V_2)$, and if $\mathrm{Val}(s)$ is not a function then it is a vector or scalar. And of course s, s_2 share the final character. Hence E1 and E2 hold.

Further, by FF–NN, $\mathrm{Val}(s)$ is a vector or vector-valued function if and only if V_1 is so. It follows that $\mathrm{Val}(s)$ is a scalar or scalar-valued function if and only if V_1 is so. And of course s, s_1 share the first character. Hence E3 and E4 hold.

Q: Let me just note that your clause NF generalizes the definition of outer product $|w\rangle\langle v|$ quoted in §1 provided that the vector spaces V and W coincide with $\mathcal{H}$.

3 Associativity

A: It turns out that parentheses are unnecessary in Dirac terms because the partial operation

$$\mathrm{Val}(s_1) * \mathrm{Val}(s_2) = \mathrm{Val}(s_1 s_2)$$

on the values of Dirac terms is associative. The operation is defined if the concatenation $s_1 s_2$ is a Dirac term, i.e. if kets and bras alternate in $s_1 s_2$.

Q: What if $\mathrm{Val}(s_i) = \mathrm{Val}(t_i)$? Will we have that $\mathrm{Val}(s_1 s_2) = \mathrm{Val}(t_1 t_2)$?

A: Yes, because the concatenation clauses in the definition of $\mathrm{Val}(s_1 s_2)$ are formulated in terms of $\mathrm{Val}(s_1)$ and $\mathrm{Val}(s_2)$, without examining the terms s_1 and s_2.

Lemma 1. *The partial operation $*$ is associative. In other words, let s_1, s_2, s_3 be Dirac terms such that kets and bras alternate in the concatenation $s_1 s_2 s_3$ and let V_1, V_2, V_3 be the values of s_1, s_2, s_3 respectively. Then*

$$(V_1 * V_2) * V_3 = V_1 * (V_2 * V_3) \tag{1}$$

Proof. First examine V_3. If V_3 is a scalar, factor it out of the equation, so that (1) becomes obvious. Similarly, if V_3 is a scalar-valued function, then

$$(V_1 * V_2) * V_3(A) = V_1 * (V_2 * V_3(A)) \qquad \text{holds for every } A \in \mathrm{Dom}(V_3) \tag{2}$$

because the scalar $V_3(A)$ can be factored out of the equation. If V_3 is a vector-valued function, it suffices to prove (2) because it implies (1). In order to prove (2) for vector-valued functions, it suffices to prove (1) for the case where V_3 is a vector. In this case, by E4, s_3 starts with ket, s_2 ends with a bra, and therefore, by E1, V_2 is a function with domain $\mathcal{H}$.

Next examine V_1. If V_1 is a scalar, factor it out, and then (1) is obvious. If V_1 is a function then, using FF and FN in §2, we have:

$$\begin{aligned}(V_1 * V_2) * V_3 &= \big(V_1 \circ V_2\big)(V_3) = V_1(V_2(V_3)) \\ V_1 * (V_2 * V_3) &= V_1 * (V_2(V_3)) = V_1(V_2(V_3))\end{aligned}$$

It remains to prove (1) in the case where V_1, V_3 are vectors and V_2 is a function with domain $\mathcal{H}$. Since V_1 is a vector, s_1 ends with a ket by E2, so that s_2 starts with a bra and V_2 is a scalar-valued function by E3. We have

$$\begin{aligned}(V_1 * V_2) * V_3 &= \big(V_1 \cdot V_2\big)(V_3) = V_1 \cdot V_2(V_3), \\ V_1 * (V_2 * V_3) &= V_1 * (V_2(V_3)) = V_1 \cdot V_2(V_3).\end{aligned}$$

□

Q: What does the associativity buy you?

A: It allows us to prove a certain robustness phenomenon which can be illustrated on the example where

$$s = \big(\langle x \,|\, y\rangle\big)\langle u|$$

Let $|v\rangle$ be an arbitrary vector in $\mathcal{H}$ and c, d be the scalar products $\langle x \,|\, y\rangle$ and $\langle u \,|\, v\rangle$, respectively. If $|y\rangle$ as is a vector ket in s then, by value, i.e., writing terms instead of their values (as it is commonly done)

$$s|v\rangle = \Big(\big(\langle x \,|\, y\rangle\big)\langle u|\Big)|v\rangle = \big(c \cdot \langle u|\big)|v\rangle = c \cdot d.$$

If $|y\rangle$ is a function ket in s then $\langle x| \circ |y\rangle$ is the the operation of multiplying by c, and so (again by value) we have

$$s|v\rangle = \Big(\langle x|\,|y\rangle\,\langle u|\Big)|v\rangle = \Big(\langle x| \circ |y\rangle \circ \langle u|\Big)|v\rangle = \Big(\langle x| \circ |y\rangle\Big) \cdot d = c \cdot d,$$

getting exactly the same result.

Lemma 2 (Robustness)**.** *Let s be a Dirac term and let a ket $|y\rangle$ occur in a particular non-final position in sequence s. $\mathrm{Val}(s)$ is the same whether (the occurrence of) $|y\rangle$ in that position is a vector ket or a function ket.*

Proof. First suppose that $|y\rangle$ is the first character in s. By Lemma 1, we may assume that s is a concatenation of $|y\rangle$ and some Dirac term s_2. Let V, V_2 be the values of $|y\rangle$ and s_2 respectively. By E3, V_2 is a scalar or a scalar-valued function. Recall that there is a vector $\vec{v}$ such that y is a marked version of $\vec{v}$. If $|y\rangle$ is a vector ket then $V = \vec{v}$, and if $|y\rangle$ is a function ket it is the function $V(a) = a \cdot \vec{v}$.

If V_2 is a scalar then, by NF–NN in §2,

$$\mathrm{Val}(s) = \begin{cases} V(V_2) = V_2 \cdot \vec{v} & \text{if } |y\rangle \text{ is a function ket,} \\ V \cdot V_2 = \vec{v} \cdot V_2 = V_2 \cdot \vec{v} & \text{if } |y\rangle \text{ is a vector ket.} \end{cases}$$

Similarly, if V_2 is a scalar-valued function then, by FF and FN, for every argument A of V_2, we have

$$\mathrm{Val}(s)(A) = \begin{cases} V(V_2(A)) = V_2(A) \cdot \vec{v} & \text{if } |y\rangle \text{ is a function ket,} \\ V \cdot V_2(A) = \vec{v} \cdot V_2(A) = V_2(A) \cdot \vec{v} & \text{if } |y\rangle \text{ is a vector ket.} \end{cases}$$

This completes the proof in the case that $|y\rangle$ is at the beginning of s.

Now suppose that $s = s_1|y\rangle s_2$ where s_1, s_2 are Dirac terms with values V_1, V_2. By Lemma 1, we may assume that s is the concatenation of s_1 and $|y\rangle s_2$, so that $\mathrm{Val}(s)$ is determined by V_1 and $\mathrm{Val}\big(|y\rangle s_2\big)$. By the first part of the proof, $\mathrm{Val}\big(|y\rangle s_2\big)$ does not depend on how the ket $|y\rangle$ is marked. It follows that $\mathrm{Val}(s)$ does not depend on how $|y\rangle$ is marked. □

Now we drop the Tentative Convention of §2. The kets are not marked anymore. One should be able to tell from the context whether a ket denotes a vector or a function.

Q: By the robustness lemma, we have a whole spectrum of possible resolutions of the abuse of notation in question.

A: One natural resolution is to view kets as function kets where possible:

In a Dirac term, an occurrence of a ket is viewed as a vector if and only if it is the final character in the term.

Q: The direct opposite strategy is to view an occurrence of a ket as a function if and only if it is the first character in the term. I'm kidding.

A: Actually, a close relative of your strategy works: View an occurrence of a ket as a function if and only if it is the first character and the last character is a bra.

Q: Explain.

A: If the first character is a bra or if the last character is a ket, then the given Dirac term has the form

$$\langle x_1|\,|y_1\rangle \ldots \langle x_n|\,|y_n\rangle, \quad \langle x_1|\,|y_1\rangle \ldots \langle x_n|\,|y_n\rangle\langle x_0|, \quad \text{or} \quad |y_0\rangle\langle x_1|\,|y_1\rangle \ldots \langle x_n|\,|y_n\rangle.$$

Pair up every $\langle x_i|$ with $|y_i\rangle$ and let $c = \prod_i \langle x_i \,|\, y_i\rangle$. Every ket is viewed as a vector, and you get c, $c \cdot \langle x_0|$, or $c \cdot |y_0\rangle$ respectively.

If the first character is a ket and the last character is a bra, the term has the form

$$|y_0\rangle\langle x_1|\,|y_1\rangle \ldots \langle x_n|\,|y_n\rangle\langle x_0|.$$

Pair up $\langle x_1|, \ldots, \langle x_n|$ with $|y_1\rangle, \ldots, |y_n\rangle$ respectively and let $c = \prod_{i=1}^{n} \langle x_i \,|\, y_i\rangle$. You get $c \cdot |y_0\rangle\langle x_0| = c \cdot |y_0\rangle \circ \langle x_0|$.

References

[1] Paul A.M. Dirac, "A new notation for quantum mechanics," *Mathematical Proceedings of the Cambridge Philosophical Society* 35:3 (1939) 416–418

[2] Michael A. Nielsen and Isaac L. Chuang, "Quantum Computation and Quantum Information," 10th Anniversary Edition, Cambridge University Press 2010

[3] Reinhard F. Werner, "Mathematical methods of quantum information theory, Lecture 1," at minute 35, https://www.youtube.com/watch?v=vb0ZEsATUcw&t=2109s

The Formal Language Theory Column

by

Giovanni Pighizzini

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

BEATCS no 141

25 EDITIONS OF DCFS: ORIGINS AND DIRECTIONS

Jürgen Dassow
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
PSF 4120, 39016 Magdeburg, Germany
dassow@iws.cs.uni-magdeburg.de

Martin Kutrib
Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
kutrib@informatik.uni-giessen.de

Giovanni Pighizzini
Dipartimento di Informatica
Università degli Studi di Milano
Via Celoria, 18, 20133 Milano, Italy
pighizzini@di.unimi.it

Abstract

Since the late nineties the scope of the *International Conference of Descriptional Complexity of Formal Systems* (DCFS) encompasses all aspects of descriptional complexity, both in theory and application. We first consider the historical development of the conference. Then we turn to some impressions from the 25 editions of the conference, which we particularly remember. In order to give a deeper inside in the field of descriptional complexity, we present some of its very basics from a general abstract perspective. Then we turn to some of the outstanding and dominating directions in the course of time. The results presented are not proved but we merely draw attention to the overall picture and some of the main ideas involved.

1 Introduction

Since the dawn of theoretical computer science the relative succinctness of different representations of (sets of) objects by formal systems have been a subject of

intensive research. An obvious choice to encode the objects is by strings over a finite alphabet. Then a set of objects is a set of strings, that is, a formal language. Formal languages can be described by several means, for example, by automata, grammars, rewriting systems, equation systems, etc. In general, such a descriptional system is a set of finite descriptors for languages. Core questions of descriptional complexity are "How succinctly (related to a size complexity measure) can a system represent a formal language in comparison with other systems?" and "What is the maximum trade-off when the representation is changed from one descriptional system to another, and can this maximum be achieved?" In the classification of automata, grammars, and related (formal) systems it turned out that the gain in economy of description heavily depends on the considered systems.

The approach to analyze the size of systems as opposed to the computational power seems to originate from Stearns [115] who studied the relative succinctness of regular languages represented by deterministic finite automata (DFAs) and deterministic pushdown automata. He showed the decidability of regularity for deterministic pushdown automata in a deep proof. The effective procedure revealed the following upper bound for the simulation. Given a deterministic pushdown automaton with $n > 1$ states and $t > 1$ stack symbols that accepts a regular language, then the number of states which is sufficient for an equivalent DFA is bounded by an expression of the order $t^{n^{n^n}}$. Later this triple exponential upper bound has been improved by one level of exponentiation in [116]. In the levels of exponentiation it is tight, as proved in [95] by obtaining a double exponential lower bound. The precise bound is still an open problem. Probably the best-known result on descriptional complexity is the construction of a DFA that simulates a given nondeterministic finite automaton (NFA) [113]. By this so-called *power-set construction*, each state of the DFA is associated with a subset of NFA states. Moreover, the construction turned out to be optimal, in general. That is, the bound on the number of states necessary for the construction is tight in the sense that for an arbitrary n there is always some n-state NFA which cannot be simulated by any DFA with strictly less than 2^n states [79, 95, 97].

Let us turn to another cornerstone of descriptional complexity theory in the seminal paper by Meyer and Fischer [95]. In general, a known upper bound for the trade-off answers the question, how succinctly can a language be represented by a descriptor of one descriptional system compared with the representation by an equivalent descriptor of the other descriptional system? In [95] the sizes of finite automata and general context-free grammars for regular languages are compared. The comparison revealed a qualitatively new phenomenon. The gain in economy of description can be arbitrary, that is, there are no recursive functions serving as upper bounds for the trade-off, which is said to be *non-recursive*. Non-recursive trade-offs usually sprout at the wayside of the crossroads of (un)decidability, and

in many cases proving such trade-offs apparently requires ingenuity and careful constructions.

Nowadays, descriptional complexity has become a large and widespread area. On our tour on the field we first consider the historical development of the conference *Descriptional Complexity of Formal Systems* (DCFS). Then we turn to some impressions from the 25 editions of the conference, which we particularly remember. In order to give a deeper inside in the field of descriptional complexity, we present some of its very basics from a general abstract perspective. Our tour on the subjects covers some outstanding and dominating topics. It obviously lacks completeness and it reflects our personal view of what constitute some of the most interesting links to descriptional complexity theory. In truth there is much more to the field than can be summarized here and in the related papers [31, 39, 69, 70]. The results presented are not proved but we merely draw attention to the overall picture and some of the main ideas involved.

2 History of DCFS [1]

In 1998, the history of DCFS started at the conference *Mathematical Foundations of Computer Science* (MFCS) in Brno. During a lunch break, Detlef Wotschke (1944–2019) suggested the organization of a workshop on descriptional complexity and related topics. It seems that there were two reasons for such a proposal.

Firstly, in 1997, within the organization *International Federation of Information Processing* (IFIP), a reestablishment of the Technical Committee TC1 *Foundations of Computer Science* took place, and within TC1 a Working Group WG 1.02 *Descriptional Complexity* was created. The chairman of this WG was Detlef Wotschke. It was natural to found a special workshop of the working group.

Secondly, the MFCS conference in Brno was accompanied by more than 10 workshops, some of them were organized as single events and some of them took place as a part of certain workshop series. Descriptional complexity was present in some of these workshops, but a special workshop on this topic was missing.

Detlef did not only come with the proposal of a workshop, he also had an idea for the place – Magdeburg (where DLT took place in 1995 and where Jürgen Dassow, the head of the group working in formal languages in Magdeburg, had a good position in the university). After some discussions, Jürgen accepted that his group will organize a workshop in Magdeburg in 1999.

In July 20–23, 1999, the workshop *Descriptional Complexity of Automata, Grammars and Related Systems* (DCAGRS) took place in Magdeburg. It was a

[1] The conference series *Descriptional Complexity of Formal Systems* has two roots, the workshops *Formal Descriptions and Software Reliability* and *Descriptional Complexity of Automata, Grammars and Related Systems*. Here we reflect only the latter one.

terrible title, but the organizers wanted a title which describes very well the topic of the workshop. The event was successful with respect to the invited lectures (e. g. J. Gruska, Sh. Yu (1950–2012), J. Shallit) as well as to the number and quality of submissions as well as to the large number of participants.

Detlef Wotschke
(1944–2019)

Helmut Jürgensen
(1942–2019)

We mention two facts where the first DCAGRS essentially differs from the later DCFS conferences. Firstly, the conference fee was only 70 euros, the average registration fee of some of the last normal DCFS conferences was 280 euros. Secondly, the program committee consisted of six person, the average of the last conferences was 24.

By the success of the first edition, it was necessary to look for a continuation. One weak before the workshop in Magdeburg, there was the *Workshop on Implementation of Automata* in Potsdam, organized by Helmut Jürgensen (1942–2019). During the excursion of this event (a boat tour through the lakes around Potsdam) Detlef and Jürgen talked to Helmut concerning the next DCAGRS. Finally, Helmut accepted to organize it in London (Ontario) (not knowing that it will be organized in London four times almost like a biannual conference, and that he will organize it three times). The site London was chosen, because, from the beginning, there was the idea to organize the workshop alternately in Europe and North America. This idea has been followed in the sequel with only three exceptions as one can see from Figure 1.

In the following years, there was a steering committee which was looking for the persons and places of the next (two) DCFS editions. This task was not easy in some cases, but finally the committee was successful in all the years. In 2015 the international workshop DCFS became an international conference to underline

the grown importance and the history of the event. A list of all editions of DCFS is given in Figure 1.

	time	place	organizing institution / chairman
DCAGRS 1999	July 20–23	Magdeburg, Germany	O.-v.-Guericke-Universität Magdeburg J. Dassow
DCAGRS 2000	July 27–29	London, Canada	The University of Western Ontario H. Jürgensen
DCAGRS 2001	July 20–22	Vienna, Austria	Technical University of Vienna R. Freund
DCFS 2002	August 21–25	London, Canada	The University of Western Ontario H. Jürgensen
DCFS 2003	July 12–14	Budapest, Hungary	Hungarian Academy of Sciences E. Csuhaj-Varjú
DCFS 2004	July 26–28	London, Canada	The University of Western Ontario L. Ilie
DCFS 2005	June 30 - July 2	Como, Italy	University of Milan G. Pighizzini
DCFS 2006	June 21–23	Las Cruces, USA	New Mexico State University H. Leung
DCFS 2007	July 20–22	Nový Smokovec, Slovakia	R.J.Šafarik University Košice V. Geffert
DCFS 2008	July 16–18	Charlottetown, Canada	University of Prince Edward Island C. Câmpeanu
DCFS 2009	July 6–9	Magdeburg, Germany	O-v-Guericke-Universität Magdeburg J. Dassow, B. Truthe
DCFS 2010	August 8-10	Saskatoon, Canada	University of Saskatchewan I. McQuillan
DCFS 2011	July 25–27	Limburg, Germany	Justus-Liebig-Universität Gießen M. Holzer, M. Kutrib
DCFS 2012	July 23–25	Braga, Portugal	Univ. of Porto and Univ. of Minho N. Moreira, R. Reis
DCFS 2013	July 22-25	London, Canada	The University of Western Ontario H. Jürgensen
DCFS 2014	August 5–8	Turku, Finland	University of Turku J. Karhumäki, A. Okhotin
DCFS 2015	July 25–27	Waterloo, Canada	University of Waterloo J. Shallit
DCFS 2016	July 5–8	Bucharest, Romania	University of Bucharest C. Câmpeanu
DCFS 2017	July 3–5	Milan, Italy	University of Milan G. Pighizzini
DCFS 2018	July 25–27	Halifax, Canada	Saint Mary's University St. Konstantinides
DCFS 2019	July 17–19	Košice, Slovakia	Slovak Academy of Sciences G. Jirásková
DCFS 2020		Collected Papers	 G. Jirásková, G. Pighizzini
DCFS 2021		Collected Papers	 Y.-S.Han, S.-K. Ko
DCFS 2022	August, 29–31	Debrecen, Hungary	University of Debrecen Gy. Vaszil
DCFS 2023	July 4–6	Potsdam, Germany	University of Potsdam H. Bordihn

Figure 1: List of conferences.

We mention some important facts.

The initiative for DCFS came from the chairman of WG 1.02 of TC1 of IFIP, and all editions were organized by some institution and this IFIP working group together. Thus DCFS can be considered as the conference of WG 1.02. Since some years, the meetings of the Working Group take place as an evening session of DCFS.

If we compare the list of topics from 2007 (the oldest which can be found in the Web) and 2023 (the last conference) and those in between, then one notice that they are identical in appr. 75% of the items. This proves that there is a strong continuity and no following of short-lived *modern* directions.

In the years 2020 and 2021, DCFS conferences were planned in Vienna organized by R. Freund and in Seoul organized by Y.-S. Han and S.-K. Ko, respectively. Due to the crisis caused by the Corona virus, both conferences had to be canceled. However, there were invitations to the world for submitting papers such that proceedings could also be published in these years. Thanks go to the editors G. Jirásková/G. Pighizzini and Y.-S. Han/S.-K. Ko for their contribution to the survival of DCFS during the Corona time.

From the very beginning there was the idea that DCAGRS/DCFS should be organized with respect time and place in connection with some other conference such that e. g., only one crossing of the Atlantic Ocean is necessary to visit at least two conferences. As favorite accompanying conferences were considered *Developments in Language Theory* (DLT) and the *International Conference Implementation and Application of Automata* (CIAA, formerly *Workshop on Implementation of Automata*, WIA). Also this idea was realized for almost all editions (see Figure 2). Sometimes, the events were very near; for instance, in 2001, there was one day which was part of the DLT as well as of the DCFS program. Sometimes, the distance was large (in 2006, the distance between Las Cruces and Santa Barbara was 1600 km, but the Europeans had to cross the ocean only once; the four days between the two conferences could be used e. g., for a visit of the Grand Canyon almost in the middle between the towns).

The special event *50 Years of Automata Theory*, that took place in 2000, was particularly remarkable. The list of speakers was very impressive. One could hear, meet and talk to all those persons which contributed by famous basic theorems as M. Rabin, D. Scott, and Sh. Greibach, introduced essential concepts as R. McNaughton or wrote famous textbooks as J. Hopcroft and A. Salomaa (note that the mentioned names represent less than half of speakers). However, we do not know why automata theory became 50 years in 2000.

In 2015, one day before DCFS, the birthday of Janusz (John) Brzozowski (1935–2019) was celebrated in a one-day-conference.

Some remarks concerning proceedings. In the years 1999–2008, proceedings were published by the organizing institution. In the following two years, the pro-

year/place	accomp. event	year/place	accomp. event
1999 Magdeburg	WIA Potsdam	2010 Saskatoon	DLT London and
2000 London	CIAA London and		CIAA Winnipeg
	50 Years Automata Th.	2012 Braga	CIAA Porto
2001 Vienna	DLT Vienna	2013 London	CIAA Halifax
2003 Budapest	DLT Szeged	2014 Turku	CIAA Gießen
2004 London	CIAA Kingston	2015 Waterloo	Birthday Brzozowski
2005 Como	CIAA Sophia Antipolis	2017 Milan	DLT Liege and
2006 Las Cruces	DLT Santa Barbara		CIAA Marne-la-Vallé
2007 Nový Smokovec	CIAA Prague	2018 Halifax	CIAA Charlottetown
2008 Charlottetown	CIAA San Francisco	2019 Košice	CIAA Košice
2009 Magdeburg	DLT Stuttgart	2022 Debrecen	NCMA Debrecen

Figure 2: List of events accompanying DCAGRS/DCFS. (NCMA is an international workshop on *Non-Classical Models of Automata and Applications*.)

ceedings appeared in the series *Electronic Proceedings in Theoretical Computer Science* as numbers 3 and 31, respectively. There were some attempts to publish in the LNCS series of Springer-Verlag, but only in 2011 we were successful. Starting with the thirteenth edition of DCFS, the Proceedings appeared as *Lecture Notes in Computer Science*.

Proceedings have mostly a page limit for the contributions, i. e., they do not contain often full versions. Therefore, from the very beginning, full versions of selected papers were published as special issues of some scientific journals. Thus,

year DCFS	journal	volume (issue)	year journal	year DCFS	journal	volume (issue)	year journal
1999	JALC	5 (3)	2000	2012	JALC	17 (2–4)	2012
2000	JALC	6 (4)	2001	2013	IJFCS	25 (7)	2014
2001	JALC	7 (4)	2002	2014	TCS	610(A)	2016
2002	JALC	9 (2–3)	2004	2015	IC	259 (2)	2018
2003	TCS	330 (2)	2005	2016	JALC	22 (1–3)	2017
2004	IJFCS	16 (5)	2005	2017	IJFCS	30 (6–7)	2019
2005	JALC	12 (1–2)	2007	2018	TCS	798	2019
2006	TCS	387 (2)	2007	2019	IC	284	2022
2007	IJFCS	19 (4)	2008	2020	JALC	28 (1–3)	2023
2008	TCS	410 (35)	2009	2021	IJFCS	in progress	
2009	JALC	15 (1–2)	2010	2022	TCS	in progress	
2010	IJFCS	23 (1)	2012	2023	IC	planned	
2011	TCS	449	2012				

Figure 3: Journal publication of selected papers.

one can find many full versions of a certain DCFS on a fixed place and not distributed over a lot of journals. For the first editions, the full versions appeared in *Journal of Automata, Languages, and Combinatorics* (JALC), a journal edited by the University of Magdeburg with persons in the editing staff, which also were involved in the program and organizing committees. Later the journals *Theoretical Computer Science* (TCS) from Elsevier B.V., *International Journal of Foundations of Computer Science* (IJFCS) from World Scientific Publishing Co., and *Information and Computation* (IC) from Elsevier B.V. were involved. The list in Figure 3 gives the journal in relation to the year of the conference.

The development of the number of accepted papers is shown in Figure 4. It is worth mentioning that the invited contributions are not included in the statistics. In most years there were additionally 4 invited presentations and papers. However, since in the early years the spirit of DCAGRS/DCFS was that of an intense workshop, at that times the number of invited speakers was higher, with a maximum of 8 speakers in 2004. Though from the very beginning all submitted papers were peer reviewed by at least three reviewers, respectively, the PCs had to work in the classical way without the support of a more or less professional conference managing system. Due to this fact, the information about the number of submitted and, thus, the number of rejected submissions is not available before 2011. The situation changed in 2011 when EasyChair came into play.

A pleasing fact is that the number of authors and their countries of affiliations has been at a good level from the beginning. This also shows that the interest in the topic has been maintained over the years and emphasizes once more that topics of descriptional complexity have a strong continuity and are not following short-lived directions. The development is shown in Figure 5.

Further information on the DCFS series (for instance programs, contents of the proceedings, special issues etc.) can be found on the web page

`http://www.informatik.uni-giessen.de/dcfs`

3 Impressions From 25 Editions of the Conference

The contents of this section consists of personal (not scientific) impressions of the first author (who did not attend all conferences such that his reflections are limited).

Mostly, the conference took place in universities or near to the universities in the towns. The exceptions were

- 2005 Como – in a theater,

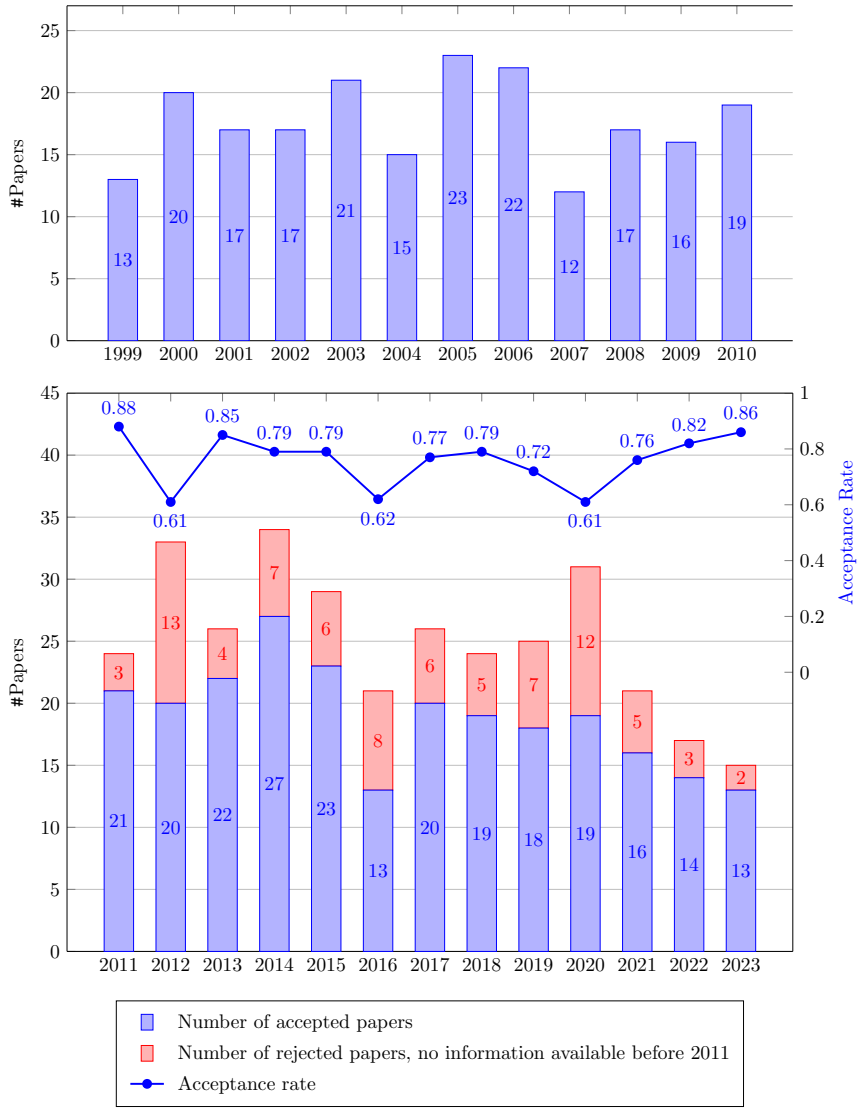


Figure 4: Development of the number of papers. Invited contributions are not included in the statistics.

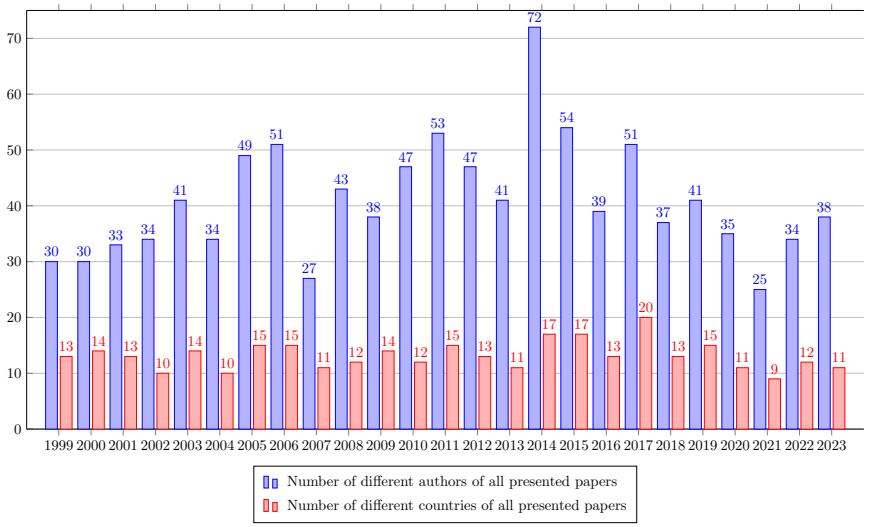


Figure 5: Development of the number of different authors and the number of different countries of all presentations. Invited contributions are included in the statistics.

- 2007 Nový Smokovec – in a hotel in the High Tatras,
- 2011 Limburg – in a hotel,
- 2012 Braga – in a museum.

If I should give the sites which impressed me most, then two places come to my remembering:

- Sala Bianca of Teatro Sociale in Como: It was a large room with wonderful baroque design which amazed me as I entered it for the first time as well as the rest of the workshop.

- Art in the University of Saskatoon: It seemed to me that the whole university was an exhibition of different arts and science. All buildings were full of sculptures, artistic installations and other works of arts, but also some terrariums. Moreover, already on the way from the hotel to the university, I saw a lot of sculptures, etc.

In connection with conferences, workshops etc., I visited a lot of places of interest. However, I remember especially the excursions of the DCFS conferences. There are two reasons for that: I have seen many sites which also contributed to my knowledge and were not only nice places to see, and I visited landscapes of extraordinary importance. Let me mention here:

Sala Bianca, Teatro Sociale, Como

University of Saskatoon

- The sites of UNESCO World Heritage
  DCFS 2012 – visit of Guimarães, where Portugal was born,
  DCFS 2017 – excursion to Bergamo (with the wall around the upper town),
  DCFS 2018 – visit of the harbour of Lunenburg,
  DCFS 2023 – excursions to the castles and gardens of Sanssouci,
  DCFS 2010 – visit of Wanuskewin, an important place for the first nations in Canada (this is not really a UNESCO site but it belongs to the Canadian proposals)

- DCFS 2006 – excursion to the desert White Sands National Park,
  DCFS 2007 – hiking tours in the High Tatras, a UNESCO Biosphere Reservat: one tour only a short walk, one tour to the mountain hut Zamkovskeho Chata (duration appr. 2 hours), and a long tour of five hours to the Téryho Chata,
  DCFS 2008 - excursion to Green Gables, a National Historic Site of Canada (it shows places related to the book *Anne of Green Gables* by L.M. Montgomery, which tells a story on a farmer girl in the 19th century and is popular in Japan, too).

Finally, some miscellaneous reminds:

In 2002, the reception took place as a barbecue in some park near London. As we reached the place, H. Jürgensen and some of his students started to encircle it with a red net used as a fence. Since there were enough place, their handling was surprising. The reason was that it is forbidden in Ontario to take alcoholic drinks at public places. However, if you have a fence and a special permission, it is allowed. Thus they ensured wine and beer for the barbecue reception.

In 2007, the long hiking tour through the High Tatras ended at the station for a funicular. Some participants took another way of return; they used scooters.

Guimarães:
first capital of Portugal

Bergamo:
wall around the upper town

However, the way was very curvy and steep in some parts such that it was a little bit dangerous to take this way. Some of the participants arrived the base station without problems, some of them had a fall of one's scooter and a painful night.

In 2010, there were a choice in the meal of the reception, lobster or something else. Because lobster is very typical for Prince Edward Island, almost all participants chose lobster. However, almost nobody had some experience in eating lobster. Therefore anybody got a bib to protect the clothes. Then everyone did his best and mostly it was done successfully.

Helmut Jürgensen and his students
inside the fence

Markus Holzer, Martin Kutrib,
Bianca Truthe, Jürgen Dassow,
and lobsters

4 Basic Concepts of Descriptional Complexity

In order to give a deeper inside in the field of descriptional complexity, we present some of its very basics from a general abstract perspective.

We denote the set of nonnegative integers by $\mathbb{N}$. Let Σ^* denote the set of all words over a finite alphabet Σ. For the *length* of a word we write $|w|$. We use $\subseteq$ for *inclusions* and $\subset$ for *strict inclusions*. In general, the family of all languages accepted by a device of some type X is denoted by $\mathscr{L}(X)$.

In order to be general, we first formalize the intuitive notion of a representation or description of a family of languages. A *descriptional system* is a collection of encodings of items where each item *represents* or *describes* a formal language. In the following, we call the items *descriptors*, and identify the encodings of some language representation with the representation itself. More precisely, a descriptional system $\mathcal{S}$ is a set of finite descriptors such that each $D \in \mathcal{S}$ describes a formal language $L(D)$. The *family of languages represented* (or *described*) by $\mathcal{S}$ is $\mathscr{L}(\mathcal{S}) = \{ L(D) \mid D \in \mathcal{S} \}$.

A *complexity measure* for a descriptional system $\mathcal{S}$ is a total recursive mapping $c\colon \mathcal{S} \to \mathbb{N}$. From the viewpoint that a descriptional system is a collection of encoding strings, the length of the strings is a natural measure for the size. We denote it by length.

For example, nondeterministic finite automata can be encoded over some fixed alphabet. The set of these encodings is a descriptional system $\mathcal{S}$, and $\mathscr{L}(\mathcal{S})$ is the family of regular languages.

Apart from length, examples for complexity measures for nondeterministic finite automata are the *number of states* and the *number of transition*.

Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be descriptional systems with complexity measures c_1 and c_2, respectively. A total function $f\colon \mathbb{N} \to \mathbb{N}$, is said to be a *lower bound* for the increase in complexity when changing from a descriptor in $\mathcal{S}_1$ to an equivalent descriptor in $\mathcal{S}_2$, if for infinitely many $D_1 \in \mathcal{S}_1$ with $L(D_1) \in \mathscr{L}(\mathcal{S}_2)$ there exists a *minimal* $D_2 \in \mathcal{S}_2(L(D_1))$ such that $c_2(D_2) \geq f(c_1(D_1))$.

A total function $f\colon \mathbb{N} \to \mathbb{N}$ is an *upper bound* for the increase in complexity when changing from a descriptor in $\mathcal{S}_1$ to an equivalent descriptor in $\mathcal{S}_2$, if for all $D_1 \in \mathcal{S}_1$ with $L(D_1) \in \mathscr{L}(\mathcal{S}_2)$, there exists a $D_2 \in \mathcal{S}_2(L(D_1))$ such that $c_2(D_2) \leq f(c_1(D_1))$.

It may happen that the upper bound is not effectively computable. If there is no recursive upper bound, then the *trade-off* for changing from a description in $\mathcal{S}_1$ to an equivalent description in $\mathcal{S}_2$ is said to be *non-recursive*. Non-recursive trade-offs are independent of particular measures. That is, whenever the trade-off from one descriptional system to another is non-recursive, one can choose an arbitrarily large recursive function f but the gain in economy of description eventually exceeds f when changing from the former system to the latter. As an

example, we consider nondeterministic pushdown automata that are used to accept regular languages. Clearly, for any such automaton there exists an equivalent finite automaton. However, the trade-off for the conversion of the pushdown automaton into the finite automaton is non-recursive.

5 Outstanding Topics

5.1 Computational Completeness with Small Resources

In the 25 edition of DCFS, there were more than 50 papers on extensions of context-free grammars (as matrix and programmed grammars etc.), insertion-deletion systems, contextual grammars, systems of grammars and automata (as Lindenmayer systems, cooperating distributed grammar systems, parallel communicating grammar systems, etc.) The problem which is mostly discussed is the following: Let a device and a numerical parameter, which describes (partly) the size of the device, be given. Let $\mathscr{L}$ be the family of languages generated by such devices. Is there a constant c such that, for each $L \in \mathscr{L}$, there is a device D which computes L and the parameter of D is at most c? Moreover, if c exists, find the minimal one.

In the sixties and seventies, a lot of variants of context-free grammars were introduced, where the sequence of the applied rules is controlled by some mechanism. We mention very informally three mechanisms and refer to [19] for details. In a *matrix grammar*, sequences of rules called matrices are given, and the generation process consists of applications of matrices, i. e., rules in the order given by the matrices; in a *programmed grammar*, with each rule, sets of successor rules are associated; in a *graph-controlled grammar*, the rules are associated to nodes of a graph and the successor rule has to be taken from the successor nodes in the graph. If one allows appearance checking, i. e., there is a set F of distinguished rules, and rules of F can be overpassed if they cannot be applied, and erasing rules, all these mentioned grammars generate all recursively enumerable languages.

As numerical parameter we take the number of nonterminals.

The first result in this direction was given by Gh. Păun in [111]. He showed that each recursively enumerable language can be generated by a matrix grammars with at most six nonterminals. An improvement was only given in 2001 in [23] and [21], where it was proved that each recursively enumerable language can be generated by a programmed grammar or a graph controlled grammar with only three nonterminals. The best known bounds where given by H. Fernau, R. Freund, M. Oswald and K. Reinhardt at DCFS'05:

Theorem 1. *(DCFS'05, [22])*

i) For each recursively enumerable language L, there is a matrix grammar with at most three nonterminals which generates L.

ii) For each recursively enumerable language L, there is a programmed grammar G with at most three nonterminals which generates L. Moreover, only two of the nonterminals are used in appearance checking mode.

iii) For each recursively enumerable language L, there is a graph controlled grammar with at most two nonterminals, both used in appearance checking mode, which generates L.

iv) The family of languages generated by graph controlled grammar with only one nonterminal used in the appearance checking mode is a proper subset of the family of all recursively enumerable languages.

We note that the results given in i) and iii) are optimal.

The operations of insertion and deletion of words are fundamental in formal language theory. They are motivated from linguistics (see contextual grammars) as well as – especially in the last 25 years – by the modeling of biological phenomena. Mostly, the insertions and deletion can only be done in a certain context, i. e., given a triple (α, w, β) of words, we can only insert w in a word x to obtain y if $x = u\alpha\beta v$ and $y = u\alpha w\beta v$ (and analogous for deletions). In context-free insertion-deletion systems, the contexts α and β are always empty. Then w can be inserted at any place in x.

A *context-free insertion-deletion system* G can be described as a 5-tuple $G = (V, T, I, D, A)$, where V and T are two alphabets with $T \subseteq V$, and $I \subseteq V^*$, $D \subseteq V^*$, and $A \subseteq V^*$ are three finite sets. The language generated by a context-free insertion-deletion system consists of all words from T^* which can be obtained from A by iterated applications of insertions of words from I and deletions of words of D.

Surprisingly, M. Margenstern, Gh. Păun, J. Rogozhin, and S. Verlan proved that already context-free insertion-deletion systems, where the length of the inserted and deleted words are short, are very powerful:

Theorem 2. *(DCFS'03, [88]) For each recursively enumerable language L, there is a context-free insertion-deletion system G where all words in I have a length at most three and all words in D have a length at most two such that G generates L.*

Two years later, S. Verlan showed that this result is optimal:

Theorem 3. *(DCFS'05, [117])*

i) A context-free insertion-deletion system, where all words of I and D have a length at most two, generates a context-free language.

ii) A context-free insertion-deletion system, where the sets I or D contain only letters, generates a context-free language.

Parallel communicating grammar systems (for short PCGSs) were introduced by Gh. Păun and L. Santean (now L. Kari) in 1989 in [112]. We only give an informal description of PCGSs. A non-returning PCGS is specified as an $(n + 3)$-tuple $G = (N, K, T, G_1, G_2, \ldots, G_n)$, where V and T are alphabets, $K = \{Q_1, Q_2, \ldots, Q_n\}$ is a set of n query symbols, and, for $1 \leq i \leq n$, $G_i = (N \cup K, T, P_i, S_i)$ is a context-free grammar with an axiom $S_i \in N$. A configuration of G is an n-tuple of words over $N \cup T \cup K$. We say that $(x_1, x_2, \ldots, x_n)$ derives in one step $(y_1, y_2, \ldots, y_n)$ if and only if

a) no x_i, $1 \leq i \leq n$, contains a query symbol, and $x_i \Longrightarrow_{G_i} y_i$ for $1 \leq i \leq n$, or

b) if $x_i = z_0 Q_{i_1} z_1 Q_{i_2} z_2 \ldots Q_{i_k} z_k$ with $z_j \in (N \cup T)^*$ for $0 \leq j \leq k$ and x_{i_j} contains no query symbol for $1 \leq j \leq k$, then $y_i = z_0 x_{i_1} z_1 x_{i_2} z_2 \ldots x_{i_k} z_k$ (i. e., query symbols are replaced by the corresponding sentential form); otherwise $y_i = x_i$.

The generated language consists of all word $x_1 \in T^*$ such that there is a configuration $(x_1, x_2, \ldots, x_n)$ which can be obtained from $(S_1, S_2, \ldots, S_n)$ by some derivation steps.

The generative power of non-returning PCGSs was open for more than 10 years. In 2000, N. Mandache proved that any recursively enumerable language can be generated by some non-returning PCGS. However, the proof allows no limitation of the number of grammars (see [87]). The first bound for the number of grammars, to generate all recursively enumerable languages was presented at DCFS'05 by Gy. Vaszil, who show that eighth grammars are sufficient. An improvement was given at DCFS'09 by E. Csuhaj-Varjú and Gy. Vaszil. Hitherto, their bound is the best known one.

Theorem 4. *(DCFS'09, [18]) For any recursively enumerable language L, there is a non-returning PCGS with n grammars which generates L.*

5.2 State Complexity of Operations

One of the most studied topics at DCFS is *operational state complexity*. The topic was presented during the first DCAGRS by Sheng Yu in this invited talk *State Complexity of Regular Languages* [120]. Dozens of papers deal with various aspects of this field. Before we start our short tour through this topic we present its basic idea.

Let $\circ$ be a fixed binary operation on languages from a family $\mathscr{L}$ that is closed under the operation. Then the $\circ$-*operation problem* can be stated as follows:

- Given a language descriptor A of size m and a language descriptor B of size n such that $L(A) \in \mathscr{L}$ and $L(B) \in \mathscr{L}$.
- Which size is sufficient and necessary in the worst-case (in terms of n and m) for a language descriptor to describe the language $L(A) \circ L(B)$?

Obviously, this problem generalizes as well to $k$ary language operations like, for example, complementation. In particular, if language descriptors are considered that are finite automata whose size is measured by their number of states, the notion *operational state complexity* is used.

Operations on (non)deterministic finite automata

First observations concerning basic operation problems of DFAs can be found in [90], where tight bounds for some operations are stated without proof. In [76] the tight bound of 2^n states for the DFA reversal was obtained in connection with Boolean automata. After the dawn the research direction on DFAs was revitalized in [121]. The systematic study of nondeterministic finite automata originates in [38].

In general, a method to obtain upper bounds for some $\circ$-operation problem is to provide an *effective procedure* that constructs a finite automaton accepting the result of the operation applied to some given finite automata. The number of states of the automaton constructed is an upper bound for the problem. To show that an upper bound is tight for all input automata to the procedure, a family of minimal automata must be given such that the resulting automata achieve that bound. These families are called *witnesses*. Naturally, a witness can also be given by a family of languages.

In [8] an automaton-independent approach, called *quotient complexity*, that is based on derivatives of languages is presented, which turned out to be a very useful technique for proving upper bounds for DFA operations (cf. [10, 11, 12]).

To give an impression of basic types of results, we provide the bounds for some basic operations on DFAs and NFAs accepting infinite general and unary regular languages in Table 1.

Subregular languages

Table 1 also reflects the distinctions between general and unary regular languages that have been made from the early beginnings. At first glance the differences between general and unary languages are not that big for NFAs while it can be exponential for DFAs. However, even if nondeterminism is available, the limitation to unary languages can have a big impact to the operational state complexity.

It turned out, that for many state complexity issues of unary languages Landau's function

$$F(n) = \max\{\, \mathrm{lcm}(x_1, \ldots, x_k) \mid x_1, \ldots, x_k \geq 1 \text{ and } x_1 + \cdots + x_k = n \,\}$$

which gives the maximal order of the cyclic subgroups of the symmetric group on n elements, plays a crucial role. Here, lcm denotes the least common multiple.

	Infinite Languages			
	NFA		DFA	
	general	unary	general	unary
$\cup$	$m+n+1$	$m+n+1$	mn	mn
$\sim$	2^n	$2^{\Theta(\sqrt{n\cdot\log n})}$	n	n
$\cap$	mn	mn	mn	mn
R	$n+1$	n	2^n	n
$\cdot$	$m+n$	$m+n-1 \leq \cdot \leq m+n$	$m2^n - t2^{n-1}$	mn
$*$	$n+1$	$n+1$	$3\cdot 2^{n-2}$	$(n-1)^2+1$
$+$	n	n		
$\setminus$			2^n-1	n
$/$			n	n

Table 1: [31] NFA and DFA state complexities for operations on infinite languages, where t is the number of accepting states of the "left" automaton, $\setminus$ denotes the left and / the right quotient by an arbitrary language. The tight lower bounds for union, intersection, and concatenation of unary DFAs require m and n to be relatively prime.

Since F depends on the irregular distribution of the prime numbers, we cannot expect to express $F(n)$ as a simple function of n. In [73, 74] the asymptotic growth rate

$$\lim_{n\to\infty}(\ln F(n)/\sqrt{n\cdot\ln n}) = 1$$

was determined, which implies the (for our purposes sufficient) rough estimate $F(n) \in 2^{\Theta(\sqrt{n\cdot\log n})}$. The connection with the complementation operation on unary languages represented by NFAs becomes evident from Table 1. This complementation is closely related to the unary NFA by DFA simulation, which causes also a state blow-up of order $F(n) \in 2^{\Theta(\sqrt{n\cdot\log n})}$. The proofs rely on a a normal form for unary NFAs introduced in [15]. It reads as follows.

Each n-state NFA over a unary alphabet can effectively be converted into an equivalent $O(n^2)$-state NFA consisting of an initial deterministic tail and some disjoint deterministic loops, where the automaton makes only a single nondeterministic decision after passing through the initial tail, which chooses one of the loops.

Apart from unary and finite languages, many other subregular language families have been considered from the viewpoint of operational state complexity. The systematic investigation of the descriptional complexity of such families origi-

nates in [5], where the state costs of determinizations are considered. The number of papers at DCFS dealing with this topic is quite huge and cannot be covered here. A comprehensive survey with valuable and detailed references is [24].

Universal witnesses

We mentioned above that the tightness of upper bounds is often shown by providing suitable witness languages. Interestingly, in [9] a witness over a ternary alphabet is obtained that shows the tightness of the DFA upper bounds for the operations union, intersection, concatenation, Kleene star, and reversal simultaneously. Therefore, it is called a *universal witness*. The universal witness is not always optimal with respect to the underlying alphabet size. However, from the state complexity view it can be seen as the most complex regular language. In particular, it can be used for even more. It is a witness for the maximal bounds on the number of atoms, the quotient complexity of atoms, the size of the syntactic semigroup, and about two dozen combined operations, where only a few require slightly modified versions of the universal witness. Further applications can be found in [13, 14].

Magic numbers

In connection with the well-known subset construction, a fundamental question was raised in [45]: Does there always exists a minimal n-state NFA whose equivalent minimal DFA has α states, for all n and α satisfying $n \leq \alpha \leq 2^n$. A number α not satisfying this condition is called a *magic number* (for n).

It was shown in [51] that no magic numbers exist for general regular languages over a ternary alphabet. For NFAs over a two-letter alphabet it was shown that $\alpha = 2^n - 2^k$ or $2^n - 2^k - 1$, for $0 \leq k \leq n/2 - 2$ [45], and $\alpha = 2^n - k$, for $5 \leq k \leq 2n - 2$ and some coprimality condition for k [46], are non-magic. In [53] it was proven that the integer α is non-magic, if $n \leq \alpha \leq 2^{\sqrt[3]{n}}$. Further non-magic numbers for a two-letter input alphabet were identified in [25] and [91].

Magic numbers for unary NFAs were studied in [26] by revising the Chrobak normal-form for NFAs. In the same paper also a brief historical summary of the magic number problem can be found. More general, magic numbers for several subregular language families were investigated in [37]. Further results on the magic number problem (in particular in relation to the operation problem on regular languages) can be found, for example, in [53, 50].

Further important sub-topics

As implied by the definition, so far, here we deal with the language operation problem in terms of worst-case complexity. However, the magic number problem can be seen as generalization. As opposed to the worst case, the range of state complexities that may result from an operation is considered. So, it is natural to look at the average case as well. In his invited talk *Size matters, but let's have it on average* at DCFS 2023, Rogério Reis considered various facets of this exciting field (see also [7, 98]).

Turning to another sub-topic, we recall that two words over a common alphabet are said to be *Parikh-equivalent* if and only if they are equal up to a permutation of their symbols or, equivalently, for each letter the number of its occurrences in the two words is the same. This notion extends in a natural way to languages, where two languages are Parikh-equivalent when for each word in the one language there is a Parikh-equivalent word in the other and vice versa. Inspired by the famous result of Parikh that each context-free language is Parikh-equivalent to some regular language [105], Parikh-equivalence has been connected to descriptional complexity issues. For example, in [75] the operational state complexity has been considered under parikh equivalence. That is, the resulting finite automaton must accept a language that is Parikh-equivalent to the precise language only.

Finally, the operation problems have been investigated not only for the devices DFA and NFA. A bunch of further devices have been considered. We mention only a few of them exemplarily. In her invited talk *Self-Verifying Finite Automata and Descriptional Complexity* at DCFS 2016, Galina Jirásková presented various aspects of descriptional complexity, including operation problems, on self-verifying finite automata [54]. See also [47] in this respect. The operation problems for two-way DFA were investigated in [56]. Alternating and Boolean automata are the devices considered, for example, in [43, 44, 52, 55, 76, 77]. The state complexity of operations on unambiguous finite automata and their languages is the main topic in [48].

5.3 Computational Models and Descriptional Complexity

A lot of work related to computation models, their descriptional complexity and other related properties has been done. More than 150 papers presented at DCFS investigate some kind of machines. It is impossible to briefly summarize and present in a complete way all the results obtained in this context. We just give some relevant examples and pointers to the literature.

While studying a computational model, the first question concerns its computational power. In the case of devices recognizing languages, this leads to investigate the class of accepted languages. The second natural question concerns the

succinctness. In particular, when a computational model can be simulated by another one, it is quite natural to investigate the cost of such a simulation in terms of the size of the descriptions. In the Introduction we already mentioned the classical example that can used to introduce descriptional complexity, namely the simulation of NFAs automata by DFAs, given by the subset construction: each one-way nondeterministic automaton with n states can be simulated by an equivalent deterministic finite automaton with 2^n states. Furthermore, it is well-known that in the worst case such a cost cannot be reduced [95].

During the DCFS conferences a lot of results have been presented concerning the costs of the relative succinctness of computational models.

Finite automata

We just mentioned the exponential cost of the simulation of NFAs by DFAs. In a paper presented at DCAGRS 1999, M. Kappes proved this cost cannot be reduced even when simulating *deterministic finite automata with multiple initial states*, i.e., if the only nondeterministic choice can be taken at the beginning of the computation, to choose the initial state in a given set [59]. This research was refined in [42] by giving an exact bound that keeps into account the cardinality of the set of possible initial states, besides the cardinality of the set of states of the simulated automaton.

Even for many non-trivial *subclasses of regular languages* (e.g., star-free languages, strictly locally testable languages) the cost of the elimination on nondeterminism remains exponential [5].

In the above mentioned results, the focus is on *one-way finite automata*, i.e., automata that scans the input tape from left to right. It is well-known that the computational power does not increase if the head can be moved in both directions, so obtaining *two-way finite automata*. A long-standing open problem related to these devices is the cost of the elimination of nondeterminism using two-way motion. This problem was formulated in 1978 by Sakoda and Sipser, who asked the costs, in terms of states, of the conversions of one-way and two-way NFAs into equivalent *two-way DFAs*, and it is still open [114]. For both questions the best-known upper bounds are exponential, while the lower bounds are polynomial. Several contributions related to this problem and, more in general, to two-way finite automata have been presented in DCFS conferences. We point out just few of them.

In his invited lecture at DCFS 2012, Ch. Kapoutsis presented *Minicomplexity* [58], a complexity theory for two-way automata which brings together several seemingly detached concepts and results.

V. Geffert and L. Isonová presented a translation of the Sakoda and Sipser question on two-way automata, to an analogous question on *pebble automata* [27].

The maximum length of the shortest string accepted by an n-state two-way finite automaton is known to be exponential in n. However, its exact value is not yet known. Recent contributions towards the solution of this problem have been presented at DCFS 2020 and 2023 [64, 89].

Extension of two-way automata, with restricted rewriting capabilities, have been also considered and they will be mentioned later.

Pushdown automata

As we just discussed, a lot of contributions related to regular languages and finite automata have been given. To represent a regular language, we could use a device from a more powerful class of machines. For instance, we could use a pushdown automaton. So the question of comparing the relative succinctness of finite automata and equivalent pushdown automata arises. This question was solved longtime ago by Meyer and Fischer by proving nonrecursive trade-offs [95].

One could ask what happens if pushdown automata in some restricted form are considered. One possible restriction is to require that the height of the pushdown store is bounded by a constant. This leads to the definition of *constant height pushdown automata*, introduced and firstly studied by Z. Bednárová, V. Geffert, C. Mereghetti, and B. Palano [29, 3]. Under this restriction, the tradeoff to finite automata is recursive. In particular, the size cost of the conversion of nondeterministic constant height pushdown automata into equivalent one-way deterministic finite automata is double exponential. So, constant height pushdown automata are very interesting for their succinctness. This line of research has been recently deepened. It is well-known that it cannot be decided whether the language accepted by a pushdown automaton is regular [95]. Notice that there exists pushdown automata that accept regular languages using a non-constant amount of pushdown store. This leads to the different question of deciding for a pushdown automaton whether there exists a constant h such that each string in the accepted language has an accepting computation using height at most h. In [110] it has been proved that also this property is undecidable. It remains undecidable when the pushdown alphabet is unary, i.e, when the machine is a one-counter automaton [109].

More in general, computations of pushdown automata have been analyzed in [6], introducing and studying *pushdown information* (roughly the properties of strings written on the pushdown store during computations of stateless pushdown automata), and in [28, 85], where the descriptional complexity of the *pushdown store language*, i.e., the set of strings that appears of the pushdown during an accepting computation, is studied. We point out that this language is regular.

A. Malcher considered *finite-turn pushdown automata*, namely pushdown au-

tomata that can switch from push to pop operations a number of time bounded by a fixed constant k, proving a series of interesting non-recursive trade-offs (e.g., from k-turns to $k+1$ turns, for each $k \geq 1$) [84].

While in the case of finite automata, having a two-way input tape does not increase the computational power, in the case of pushdown automata the situation is different. In fact, *two-way pushdown automata* can recognize even non-context-free languages. At the moment is still unknown if these devices are able to recognize all context-sensitive languages. In [86] the authors started an investigation on these devices in the case of a constant number of head reversals.

Questions related to *input driven pushdown automata* (also known as *nested word* or *visibly pushdown automata*) have been investigated in [106, 100, 36].

In [40] the authors consider *one-time nondeterministic* finite and pushdown automata. In these devices, whenever a guess is performed, it remains fixed for the rest of the computation. In the case of finite automata, the state increase to equivalent deterministic devices is bounded by an exponential function, while in the case of pushdown automata nonrecursive size trade-offs have been proved.

Turing machines and their variants, Cellular automata

The model of *restarting automata* has been the subject of many papers presented at DCFS and in other related conferences. This is a formal model for the *analysis by reduction*, which is used in linguistic to analyze sentences of natural languages. Roughly, this technique consists in a stepwise simplification of a given sentence in such a way that the syntactical correctness or incorrectness of the sentence is not affected. Such a process can be modeled by machines having a flexible tape where, at some point, such a simplification is performed and then the computation is restarted.

In his invited lecture *On Restarting Automata with Window Size One* at DCFS 2011, F. Otto presented a general overview of the most important variants of restarting automata, together with new results on restarting automata with window size one [101]. Several other contributions in this area have been presented in DCFS conferences. Among them, we address the reader to [41, 72, 102, 103].

Turing machines with restricted rewriting capabilities have been considered in several papers. At DCFS 2005, B. Durak presented *worm automata*. These devices are two-way finite automata equipped with a write-once track. In spite of this possibility, they still recognize only regular languages [20].

More recently, several results on *limited automata* have been presented. These devices are single-tape Turing machines in which the content of each tape cell can be rewritten only in the first d visits, for a fixed integer $d \geq 0$. In case $d \leq 1$

these devices accept only regular languages, while for each fixed $d > 1$, they characterize the class of context-free languages. The conversion from nondeterministic 1-limited automata into equivalent one-way deterministic finite automata costs, in the worst case, double exponential in size. So these devices can be extremely succinct [108]. Other results on the descriptional complexity of limited automata have presented in [71, 34]. A survey on limited automata has been given in the invited lecture *Limited Automata: Properties, Complexity and Variants* by G. Pighizzini at DCFS 2019 [107].

In his invited lecture *The Descriptional Power of Sublogarithmic Resource Bounded Turing Machines* at DCFS 2007, C. Mereghetti presented a complete picture of lower bounds on space and input head reversals for deterministic, nondeterministic, and alternating Turing machines accepting nonregular languages [92].

In [66], M. Kutrib investigated multitape Turing machines having a restricted number of nondeterministic steps, proving the existence of a nondeterministic language hierarchy between real time and linear time.

Among other computational models, it is also suitable to mention *cellular automata*. They have been the subject of invited lectures *Cellular Automata and Descriptional Complexity* by A. Malcher at DCFS 2006 [83] and *Linear Algebra Based Bounds for One-Dimensional Cellular Automat* by J. Kari at DCFS 2011 [60]. The descriptional complexity and the properties of several variants of cellular automata (e.g., one-way and two-way) have been the subject of various talks (e.g [80, 81, 82, 68]).

Non classical computation modes: probabilistic and quantum

Besides classical modes of computations, mainly based on determinism, nondeterminism, and alternation, several contributions of other modes have been presented.

At DCFS 2009, Ch. Baier gave the invited talk *Probabilistic Automata over Infinite Words: Expressiveness, Efficiency, and Decidability* [2]. Probabilistic models have been also considered in [96, 49].

Quantum automata and quantum computations have been the subject of invited talks *Descriptional complexity issues in quantum computing* and *Succinctness in quantum information processing* by J. Gruska at DCAGRS 1999 and DCFS 2003 [32, 33], *Some formal tools for analyzing quantum automata* by A. Bertoni at DCFS 2005 [4], and *Recent Developments in Quantum Algorithms and Complexity* A. Ambainis at DCFS 2014 [1]. Several contributions to this area have been presented [93, 118, 119]. In [94] the authors compare the succinctness of deterministic, nondeterministic, probabilistic and quantum finite automata.

Non-recursive trade-offs

A general survey on non-recursive trade-offs, with a unifying approach to the proof of them, has been given by M. Kutrib in his invited talk *The phenomenon of non-recursive trade-offs* at DCFS 2004 [67]. Further developments on this phenomenon have been obtained in [30].

In [57], Ch. Kapoutsis presented non-recursive trade-offs for multi-head two-way finite automata and multi-counter automata. We already mentioned the paper by A. Malcher with non-recursive trade-offs for related to finite-turn pushdown automata [84]. In many other papers (e.g. related to pushdown automata [86, 40, 109]) results presenting non-recursive trade-offs have been given.

Ambiguity and measures of nondeterminism

In this invited talk *Descriptional complexity of nfa of different ambiguity* at DCFS 2004, H. Leung presented relationships between descriptional complexity and ambiguity degree for nondeterministic finite automata [78]. Further results on this topic are given in [65]. The case of Büchi automata was considered in [99],

Unambiguity in automata theory was the title of the invited lecture given by Th. Colcombet at DCFS 2015 [17]: the concept of unambiguity, seen as a generalization of determinism, has been explored not only in automata on finite words, but in some extensions of them as, e.g., automata on infinite trees and tropical automata.

In [104] various *measure of nondeterminism* for finite automata have been investigated and compared. This idea was further explored in some subsequent papers. Among them we mention [61, 63]. More recently, also measures for *alternating automata* have been introduced and studied [62, 35]

References

[1] Ambainis, A.: Recent developments in quantum algorithms and complexity. In: Descriptional Complexity of Formal Systems (DCFS 2014). LNCS, vol. 8614, pp. 1–4. Springer (2014). https://doi.org/10.1007/978-3-319-09704-6_1

[2] Baier, C., Bertrand, N., Größer, M.: Probabilistic automata over infinite words: Expressiveness, efficiency, and decidability. In: Descriptional Complexity of Formal Systems (DCFS 2009). EPTCS, vol. 3, pp. 3–16 (2009). https://doi.org/10.4204/EPTCS.3.1

[3] Bednárová, Z., Geffert, V., Mereghetti, C., Palano, B.: Removing nondeterminism in constant height pushdown automata. Inf. Comput. **237**, 257–267 (2014). https://doi.org/10.1016/j.ic.2014.03.002

[4] Bertoni, A., Mereghetti, C., Palano, B.: Some formal tools for analyzing quantum automata. Theor. Comput. Sci. **356**, 14–25 (2006). https://doi.org/10.1016/j.tcs.2006.01.042

[5] Bordihn, H., Holzer, M., Kutrib, M.: Determinization of finite automata accepting subregular languages. Theor. Comput. Sci. **410**, 3209–3222 (2009). https://doi.org/10.1016/j.tcs.2009.05.019

[6] Bordihn, H., Jürgensen, H.: Pushdown information. In: Descriptional Complexity of Formal Systems (DCFS 2004). pp. 111–120. Report No. 619, Department of Computer Science, The University of Western Ontario, Canada (2004)

[7] Broda, S., Machiavelo, A., Moreira, N., Reis, R.: Analytic combinatorics and descriptional complexity of regular languages on average. SIGACT News **51**, 38–56 (2020). https://doi.org/10.1145/3388392.3388401

[8] Brzozowski, J.A.: Quotient complexity of regular languages. J. Autom. Lang. Comb. **15**, 71–89 (2010). https://doi.org/10.25596/jalc-2010-071

[9] Brzozowski, J.A.: In search of the most complex regular language. Int. J. Found. Comput. Sci. **24**, 692–708 (2013). https://doi.org/10.1142/S0129054113400133

[10] Brzozowski, J.A., Jirásková, G., Li, B.: Quotient complexity of ideal languages. Theor. Comput. Sci. **470**, 36–52 (2013). https://doi.org/10.1016/j.tcs.2012.10.055

[11] Brzozowski, J.A., Jirásková, G., Zou, C.: Quotient complexity of closed languages. Theor. Comput. Sci. **54**, 277–292 (2014). https://doi.org/10.1007/s00224-013-9515-7

[12] Brzozowski, J.A., Liu, B.: Quotient complexity of star-free languages. Int. J. Found. Comput. Sci. **23**, 1261–1276 (2012). https://doi.org/10.1142/S0129054112400515

[13] Brzozowski, J.A., Liu, D.: Universal witnesses for state complexity of basic operations combined with reversal. In: Implementation and Application of Automata (CIAA 2013). LNCS, vol. 7982, pp. 72–83. Springer (2013). https://doi.org/10.1007/978-3-642-39274-0_8

[14] Brzozowski, J.A., Liu, D.: Universal witnesses for state complexity of Boolean operations and concatenation combined with star. In: Descriptional Complexity of Formal Systems (DCFS 2013). LNCS, vol. 8031, pp. 30–41. Springer (2013). https://doi.org/10.1007/978-3-642-39310-5_5

[15] Chrobak, M.: Finite automata and unary languages. Theor. Comput. Sci. **47**, 149–158 (1986). https://doi.org/10.1016/0304-3975(86)90142-8, errata: [16]

[16] Chrobak, M.: Errata to "finite automata and unary languages". Theor. Comput. Sci. **302**, 497–498 (2003). https://doi.org/10.1016/S0304-3975(03)00136-1

[17] Colcombet, T.: Unambiguity in automata theory. In: Descriptional Complexity of Formal Systems (DCFS 2015). LNCS, vol. 9118, pp. 3–18. Springer (2015). https://doi.org/10.1007/978-3-319-19225-3_1

[18] Csuhaj-Varjú, E., Vaszil, G.: On the descriptional complexity of context-free non-returning pc grammar systems. J. Autom. Lang. Comb. **15**, 91–105 (2010). https://doi.org/10.25596/jalc-2010-091

[19] Dassow, J., Păun, G.: Regulated Rewriting in Formal Language Theory. Springer (1989)

[20] Durak, B.: Two-way finite automata with a write-once track. J. Autom. Lang. Comb. **12**(1-2), 97–115 (2007). https://doi.org/10.25596/jalc-2007-097

[21] Fernau, H.: Nonterminal complexity of programmed grammars. In: Machines, Computations, and Universality (MCU 2001). LNCS, vol. 2055, pp. 202–213. Springer (2001). https://doi.org/10.1007/3-540-45132-3_13

[22] Fernau, H., Freund, R., Oswald, M., Reinhardt, K.: Refining the nonterminal complexity of graph-controlled, programmed, and matrix grammars. J. Autom. Lang. Comb. **12**, 117–138 (2007). https://doi.org/10.25596/jalc-2007-117

[23] Freund, R., Păun, G.: On the number of nonterminal symbols in graph-controlled, programmed, and matrix grammars. In: Machines, Computations, and Universality (MCU 2001). LNCS, vol. 2055, pp. 214–225. Springer (2001). https://doi.org/10.1007/3-540-45132-3_14

[24] Gao, Y., Moreira, N., Reis, R., Yu, S.: A survey on operational state complexity. J. Autom. Lang. Comb. **21**, 251–310 (2016). https://doi.org/10.25596/jalc-2016-251

[25] Geffert, V.: (Non)determinism and the size of one-way finite automata. In: Descriptional Complexity of Formal Systems (DCFS 2005). pp. 23–37. Rapporto Tecnico 06-05, Università degli Studi di Milano (2005)

[26] Geffert, V.: Magic numbers in the state hierarchy of finite automata. Inform. Comput. **205**, 1652–1670 (2007). https://doi.org/10.1016/j.ic.2007.07.001

[27] Geffert, V., Istonová, L.: Translation from classical two-way automata to pebble two-way automata. RAIRO Theor. Informatics Appl. **44**, 507–523 (2010). https://doi.org/10.1051/ita/2011001

[28] Geffert, V., Malcher, A., Meckel, K., Mereghetti, C., Palano, B.: A direct construction of finite state automata for pushdown store languages. In: Descriptional Complexity of Formal Systems (DCFS 2013). LNCS, vol. 8031, pp. 90–101. Springer (2013). https://doi.org/10.1007/978-3-642-39310-5_10

[29] Geffert, V., Mereghetti, C., Palano, B.: More concise representation of regular languages by automata and regular expressions. Inf. Comput. **208**, 385–394 (2010). https://doi.org/10.1016/j.ic.2010.01.002

[30] Gruber, H., Holzer, M., Kutrib, M.: On measuring non-recursive trade-offs. J. Autom. Lang. Comb. **15**, 107–120 (2010). https://doi.org/10.25596/jalc-2010-107

[31] Gruber, H., Holzer, M., Kutrib, M.: Descriptional complexity of regular languages. In: Handbook of Automata Theory, pp. 411–457. European Mathematical Society Publishing House (2021). https://doi.org/10.4171/Automata-1/12

[32] Gruska, J.: Descriptional complexity issues in quantum computing. J. Autom. Lang. Comb. **5**, 191–218 (2000). https://doi.org/10.25596/jalc-2000-191

[33] Gruska, J.: Succinctness in quantum information processing. In: Descriptional Complexity of Formal Systems (DCFS 2003). pp. 15–25. MTA SZTAKI, Hungarian Academy of Sciences (2003)

[34] Guillon, B., Prigioniero, L.: Linear-time limited automata. Theor. Comput. Sci. **798**, 95–108 (2019). https://doi.org/10.1016/j.tcs.2019.03.037

[35] Han, Y., Kim, S., Ko, S., Salomaa, K.: Existential and universal width of alternating finite automata. In: Descriptional Complexity of Formal Systems (DCFS 2023). LNCS, vol. 13918, pp. 51–64. Springer (2023). https://doi.org/10.1007/978-3-031-34326-1_4

[36] Han, Y., Ko, S., Salomaa, K.: Limited nondeterminism of input-driven pushdown automata: Decidability and complexity. In: Descriptional Complexity of Formal Systems (DCFS 2019). LNCS, vol. 11612, pp. 158–170. Springer (2019). https://doi.org/10.1007/978-3-030-23247-4_12

[37] Holzer, M., Jakobi, S., Kutrib, M.: The magic number problem for subregular language families. Int. J. Found. Comput. Sci. **23**, 115–131 (2012). https://doi.org/10.1142/S0129054112400084

[38] Holzer, M., Kutrib, M.: Nondeterministic descriptional complexity of regular languages. Int. J. Found. Comput. Sci. **14**, 1087–1102 (2003). https://doi.org/10.1142/S0129054103002199

[39] Holzer, M., Kutrib, M.: Descriptional complexity – An introductory survey. In: Scientific Applications of Language Methods, pp. 1–58. Imperial College Press (2010)

[40] Holzer, M., Kutrib, M.: One-time nondeterministic computations. Int. J. Found. Comput. Sci. **30**, 1069–1089 (2019). https://doi.org/10.1142/S012905411940029X

[41] Holzer, M., Kutrib, M., Reimann, J.: Non-recursive trade-offs for deterministic restarting automata. J. Autom. Lang. Comb. **12**, 195–213 (2007). https://doi.org/10.25596/jalc-2007-195

[42] Holzer, M., Salomaa, K., Yu, S.: On the state complexity of k-entry deterministic finite automata. J. Autom. Lang. Comb. **6**, 453–466 (2001). https://doi.org/10.25596/jalc-2001-453

[43] Hospodár, M., Jirásková, G.: The complexity of concatenation on deterministic and alternating finite automata. RAIRO Inform. Théor. **52**, 153–168 (2018). https://doi.org/10.1051/ita/2018011

[44] Hospodár, M., Jirásková, G., Krajnáková, I.: Operations on Boolean and alternating finite automata. In: Computer Science Symposium in Russia (CSR 2018). LNCS, vol. 10846, pp. 181–193. Springer (2018). https://doi.org/10.1007/978-3-319-90530-3_16

[45] Iwama, K., Kambayashi, Y., Takaki, K.: Tight bounds on the number of states of DFAs that are equivalent to n-state NFAs. Theor. Comput. Sci. **237**, 485–494 (2000). https://doi.org/10.1016/S0304-3975(00)00029-3

[46] Iwama, K., Matsuura, A., Paterson, M.: A family of NFAs which need $2^n - \alpha$ deterministic states. Theor. Comput. Sci. **301**, 451–462 (2003). https://doi.org/10.1016/S0304-3975(02)00891-5

[47] Jirásek, J.S., Jirásková, G., Szabari, A.: Operations on self-verifying finite automata. In: Computer Science Symposium in Russia (CSR 2015). LNCS, vol. 9139, pp. 231–261. Springer (2015). https://doi.org/10.1007/978-3-319-20297-6_16

[48] Jirásek Jr., J., Jirásková, G., Sebej, J.: Operations on unambiguous finite automata. Int. J. Found. Comput. Sci. **29**, 861–876 (2018). https://doi.org/10.1142/S012905411842008X

[49] Jirásková, G.: Note on the complexity of Las Vegas automata problems. RAIRO Theor. Informatics Appl. **40**, 501–510 (2006). https://doi.org/10.1051/ita:2006033

[50] Jirásková, G.: Concatenation of regular languages and descriptional complexity. Theor. Comput. Sci. **49**, 306–318 (2011). https://doi.org/10.1007/s00224-011-9318-7

[51] Jirásková, G.: Magic numbers and ternary alphabet. Int. J. Found. Comput. Sci. **22**, 331–344 (2011). https://doi.org/10.1142/S0129054111008076

[52] Jirásková, G.: Descriptional complexity of operations on alternating and Boolean automata. In: Computer Science Symposium in Russia (CSR 2012). LNCS, vol. 7353, pp. 196–204. Springer (2012). https://doi.org/10.1007/978-3-642-30642-6_19

[53] Jirásková, G.: The ranges of state complexities for complement, star, and reversal of regular languages. Int. J. Found. Comput. Sci. **25**, 101 (2014). https://doi.org/10.1142/S0129054114500063

[54] Jirásková, G.: Self-verifying finite automata and descriptional complexity. In: Descriptional Complexity of Formal Systems (DCFS 2016). LNCS, vol. 9777, pp. 29–44. Springer (2016). https://doi.org/10.1007/978-3-319-41114-9_3

[55] Jirásková, G., Krajnáková, I.: Square on deterministic, alternating, and Boolean finite automata. Int. J. Found. Comput. Sci. **30**, 1117–1134 (2019). https://doi.org/10.1142/S0129054119400318

[56] Jirásková, G., Okhotin, A.: On the state complexity of operations on two-way finite automata. In: Developments in Language Theory (DLT 2008). LNCS, vol. 5257, pp. 443–454. Springer (2008). https://doi.org/10.1007/978-3-540-85780-8_35

[57] Kapoutsis, C.A.: Non-recursive trade-offs for two-way machines. Int. J. Found. Comput. Sci. **16**, 943–956 (2005). https://doi.org/10.1142/S012905410500339X

[58] Kapoutsis, C.A.: Minicomplexity. J. Autom. Lang. Comb. **17**, 205–224 (2012). https://doi.org/10.25596/jalc-2012-205

[59] Kappes, M.: Descriptional complexity of deterministic finite automata with multiple initial states. J. Autom. Lang. Comb. **5**, 269–278 (2000). https://doi.org/10.25596/jalc-2000-269

[60] Kari, J.: Linear algebra based bounds for one-dimensional cellular automata. In: Descriptional Complexity of Formal Systems (DCFS 2011). LNCS, vol. 6808, pp. 1–7. Springer (2011). https://doi.org/10.1007/978-3-642-22600-7_1

[61] Keeler, C., Salomaa, K.: Branching measures and nearly acyclic NFAs. Int. J. Found. Comput. Sci. **30**, 1135–1155 (2019). https://doi.org/10.1142/S012905411940032X

[62] Keeler, C., Salomaa, K.: Width measures of alternating finite automata. In: Descriptional Complexity of Formal Systems (DCFS 2021). LNCS, vol. 13037, pp. 88–99. Springer (2021). https://doi.org/10.1007/978-3-030-93489-7_8

[63] Keeler, C., Salomaa, K.: Structural properties of NFAs and growth rates of nondeterminism measures. Inf. Comput. **284**, 104690 (2022). https://doi.org/10.1016/j.ic.2021.104690

[64] Krymski, S., Okhotin, A.: Longer shortest strings in two-way finite automata. In: Descriptional Complexity of Formal Systems (DCFS 2020). LNCS, vol. 12442, pp. 104–116. Springer (2020). https://doi.org/10.1007/978-3-030-62536-8_9

[65] Kupke, J.: A powerful tool in lower-bounding constantly ambiguous automata. In: Descriptional Complexity of Formal Systems (DCFS 2006). pp. 276–284. New Mexico State University, Las Cruces, New Mexico, USA (2006)

[66] Kutrib, M.: Refining nondeterminism below linear time. J. Autom. Lang. Comb. **7**, 533–547 (2002). https://doi.org/10.25596/jalc-2002-533

[67] Kutrib, M.: The phenomenon of non-recursive trade-offs. Int. J. Found. Comput. Sci. **16**, 957–973 (2005). https://doi.org/10.1142/S0129054105003406

[68] Kutrib, M., Malcher, A.: One-way cellular automata, bounded languages, and minimal communication. J. Autom. Lang. Comb. **15**, 135–153 (2010). https://doi.org/10.25596/jalc-2010-135

[69] Kutrib, M., Moreira, N., Pighizzini, G., Reis, R.: Hot current topics of descriptional complexity. In: Advancing Research in Information and Communication Technology – IFIP's Exciting First 60+ Years, IFIP Advances in Information and Communication Technology, vol. 600, pp. 3–28. Springer (2021). https://doi.org/10.1007/978-3-030-81701-5_1

[70] Kutrib, M., Pighizzini, G.: Recent trends in descriptional complexity of formal languages. Bull. EATCS **111** (2013)

[71] Kutrib, M., Pighizzini, G., Wendlandt, M.: Descriptional complexity of limited automata. Inf. Comput. **259**, 259–276 (2018). https://doi.org/10.1016/j.ic.2017.09.005

[72] Kutrib, M., Reimann, J.: Optimal simulations of weak restarting automata. Int. J. Found. Comput. Sci. **19**, 795–811 (2008). https://doi.org/10.1142/S0129054108005966

[73] Landau, E.: Über die Maximalordnung der Permutationen gegebenen Grades. Archiv der Math. und Phys. **3**, 92–103 (1903)

[74] Landau, E.: Handbuch der Lehre von der Verteilung der Primzahlen. Teubner, Leipzig (1909)

[75] Lavado, G.J., Pighizzini, G., Seki, S.: Operational state complexity under Parikh equivalence. In: Descriptional Complexity of Formal Systems (DCFS 2014). LNCS, vol. 8614, pp. 294–305. Springer (2014). https://doi.org/10.1007/978-3-319-09704-6_26

[76] Leiss, E.L.: Succinct representation of regular languages by Boolean automata. Theor. Comput. Sci. **13**, 323–330 (1981). https://doi.org/10.1016/S0304-3975(81)80005-9

[77] Leiss, E.L.: Succinct representation of regular languages by Boolean automata. II. Theor. Comput. Sci. **38**, 133–136 (1985). https://doi.org/10.1016/0304-3975(85)90215-4

[78] Leung, H.: Descriptional complexity of nfa of different ambiguity. Int. J. Found. Comput. Sci. **16**, 975–984 (2005). https://doi.org/10.1142/S0129054105003418

[79] Lupanov, O.B.: A comparison of two types of finite sources. Problemy Kybernetiki **9**, 321–326 (1963), (in Russian), German translation: Über den Vergleich zweier Typen endlicher Quellen. Probleme der Kybernetik 6 (1966), 328–335

[80] Malcher, A.: Descriptional complexity of cellular automata and decidability questions. J. Autom. Lang. Comb. **7**, 549–560 (2002). https://doi.org/10.25596/jalc-2002-549

[81] Malcher, A.: On one-way cellular automata with a fixed number of cells. Fundam. Inform. **58**, 355–368 (2003)

[82] Malcher, A.: On two-way communication in cellular automata with a fixed number of cells. Theor. Comput. Sci. **330**, 325–338 (2005). https://doi.org/10.1016/j.tcs.2004.04.014

[83] Malcher, A.: Cellular automata and descriptional complexity. In: Descriptional Complexity of Formal Systems (DCFS 2006). pp. 26–40. New Mexico State University, Las Cruces, New Mexico, USA (2006)

[84] Malcher, A.: On recursive and non-recursive trade-offs between finite-turn pushdown automata. J. Autom. Lang. Comb. **12**, 265–277 (2007). https://doi.org/10.25596/jalc-2007-265

[85] Malcher, A., Meckel, K., Mereghetti, C., Palano, B.: Descriptional complexity of pushdown store languages. J. Autom. Lang. Comb. **17**, 225–244 (2012). https://doi.org/10.25596/jalc-2012-225

[86] Malcher, A., Mereghetti, C., Palano, B.: Descriptional complexity of two-way pushdown automata with restricted head reversals. Theor. Comput. Sci. **449**, 119–133 (2012). https://doi.org/10.1016/j.tcs.2012.04.007

[87] Mandache, N.: On the computational power of context-free pc grammar systems. Theor. Comput. Sci. **237**, 135–148 (2000). https://doi.org/10.1016/S0304-3975(98)00159-5

[88] Margenstern, M., Păun, G., Rogozhin, Y., Verlan, S.: Context-free insertion-deletion systems. Theor. Comput. Sci. **330**, 317–328 (2005). https://doi.org/10.1016/j.tcs.2004.06.031

[89] Martynova, O., Okhotin, A.: Shortest accepted strings for two-way finite automata: Approaching the 2^n lower bound. In: Descriptional Complexity of Formal Systems (DCFS 2023). LNCS, vol. 13918, pp. 134–145. Springer (2023). https://doi.org/10.1007/978-3-031-34326-1_10

[90] Maslov, A.N.: Estimates of the number of states of finite automata. Soviet Math. Dokl. **11**, 1373–1375 (1970), (English translation), in Russian: Dokl. Akad. Nauk SSSR 194 (1970), 1266–1268

[91] Matsuura, A., Saito, Y.: Equivalent transformation of minimal finite automata over a two-letter alphabet. In: Descriptional Complexity of Formal Systems (DCFS 2008). pp. 224–232. University of Prince Edward Island, Canada (2008)

[92] Mereghetti, C.: Testing the descriptional power of small Turing machines on nonregular language acceptance. Int. J. Found. Comput. Sci. **19**, 827–843 (2008). https://doi.org/10.1142/S012905410800598X

[93] Mereghetti, C., Palano, B.: Quantum automata for some multiperiodic languages. Theor. Comput. Sci. **387**, 177–186 (2007). https://doi.org/10.1016/j.tcs.2007.07.037

[94] Mereghetti, C., Palano, B., Pighizzini, G.: Note on the succinctness of deterministic, nondeterministic, probabilistic and quantum finite automata. RAIRO Theor. Informatics Appl. **35**, 477–490 (2001). https://doi.org/10.1051/ita:2001106

[95] Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: Symposium on Switching and Automata Theory (SWAT 1971). pp. 188–191. IEEE (1971). https://doi.org/10.1109/SWAT.1971.11

[96] Milani, M., Pighizzini, G.: Tight bounds on the simulation of unary probabilistic automata by deterministic automata. J. Autom. Lang. Comb. **6**, 481–492 (2001). https://doi.org/10.25596/jalc-2001-481

[97] Moore, F.R.: On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. IEEE Trans. Comput. **20**, 1211–1214 (1971). https://doi.org/10.1109/T-C.1971.223108

[98] Nicaud, C.: Average state complexity of operations on unary automata. In: Mathematical Foundations of Computer Science (MFCS 1999). LNCS, vol. 1672, pp. 231–240. Springer (1999). https://doi.org/10.1007/3-540-48340-3_21

[99] Nießner, F.: Büchi automata and their degrees of nondeterminism and ambiguity. J. Autom. Lang. Comb. **9**, 347–363 (2004). https://doi.org/10.25596/jalc-2004-347

[100] Okhotin, A., Salomaa, K.: Further closure properties of input-driven pushdown automata. Theor. Comput. Sci. **798**, 65–77 (2019). https://doi.org/10.1016/j.tcs.2019.04.006

[101] Otto, F.: On restarting automata with window size one. In: Descriptional Complexity of Formal Systems (DCFS 2011). LNCS, vol. 6808, pp. 8–33. Springer (2011). https://doi.org/10.1007/978-3-642-22600-7_2

[102] Otto, F.: On the descriptional complexity of deterministic ordered restarting automata. In: Descriptional Complexity of Formal Systems (DCFS 2014). LNCS, vol. 8614, pp. 318–329. Springer (2014). https://doi.org/10.1007/978-3-319-09704-6_28

[103] Otto, F., Kwee, K.: On the descriptional complexity of stateless deterministic ordered restarting automata. Inf. Comput. **259**, 277–302 (2018). https://doi.org/10.1016/j.ic.2017.09.006

[104] Palioudakis, A., Salomaa, K., Akl, S.G.: Comparisons between measures of nondeterminism on finite automata. In: Descriptional Complexity of Formal Systems (DCFS 2013). LNCS, vol. 8031, pp. 217–228. Springer (2013). https://doi.org/10.1007/978-3-642-39310-5_21

[105] Parikh, R.J.: On context-free languages. J. ACM **13**, 570–581 (1966). https://doi.org/10.1145/321356.321364

[106] Piao, X., Salomaa, K.: Operational state complexity of nested word automata. Theor. Comput. Sci. **410**, 3290–3302 (2009). https://doi.org/10.1016/j.tcs.2009.05.002

[107] Pighizzini, G.: Limited automata: Properties, complexity and variants. In: Descriptional Complexity of Formal Systems (DCFS 2019). LNCS, vol. 11612, pp. 57–73. Springer (2019). https://doi.org/10.1007/978-3-030-23247-4_4

[108] Pighizzini, G., Pisoni, A.: Limited automata and regular languages. Int. J. Found. Comput. Sci. **25**, 897–916 (2014). https://doi.org/10.1142/S0129054114400140

[109] Pighizzini, G., Prigioniero, L.: Pushdown and one-counter automata: Constant and non-constant memory usage. In: Descriptional Complexity of Formal Systems (DCFS 2023). LNCS, vol. 13918, pp. 146–157. Springer (2023). https://doi.org/10.1007/978-3-031-34326-1_11

[110] Pighizzini, G., Prigioniero, L.: Pushdown automata and constant height: decidability and bounds. Acta Informatica **60**, 123–144 (2023). https://doi.org/10.1007/s00236-022-00434-0

[111] Păun, G.: Six nonterminals are enough for generating each r.e. language by a matrix grammar. Internat. J. Comput. Math. **15**, 23–37 (1984). https://doi.org/10.1080/00207168408803399

[112] Păun, G., Santean, L.: Parallel communicating grammar systems: the regular case. Ann. Univ. Buc., Ser. Matem.-Inform. **38**, 55–63 (1989)

[113] Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM J. Res. Dev. **3**, 114–125 (1959). https://doi.org/10.1147/rd.32.0114

[114] Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two way finite automata. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing (STOC 1978). pp. 275–286. ACM, ACM Press (1978). https://doi.org/10.1145/800133.804357

[115] Stearns, R.E.: A regularity test for pushdown machines. Inform. Control **11**, 323–340 (1967). https://doi.org/10.1016/S0019-9958(67)90591-8

[116] Valiant, L.G.: Regularity and related problems for deterministic pushdown automata. J. ACM **22**, 1–10 (1975). https://doi.org/10.1145/321864.321865

[117] Verlan, S.: On minimal context-free insertion-deletion systems. J. Autom. Lang. Comb. **12**, 317–328 (2007). https://doi.org/10.25596/jalc-2007-317

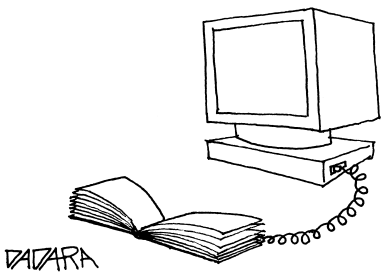
[118] Villagra, M., Yamakami, T.: Quantum state complexity of formal languages. In: Descriptional Complexity of Formal Systems (DCFS 2015). LNCS, vol. 9118, pp. 280–291. Springer (2015). https://doi.org/10.1007/978-3-319-19225-3_24

[119] Yamakami, T.: How does adiabatic quantum computation fit into quantum automata theory? Inf. Comput. **284**, 104694 (2022). https://doi.org/10.1016/j.ic.2021.104694

[120] Yu, S.: State complexity of regular languages. In: Dassow, J., Wotschke, D. (eds.) International Workshop on Descriptional Complexity of Automata, Grammars and Related Structures, Magdeburg, Germany, July 20 - 23, 1999. Preproceedings. vol. Preprint Nr. 17, pp. 77–88. Fakultät für Informatik, Universität Magdeburg, Magdeburg, Germany (1999)

[121] Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theor. Comput. Sci. **125**, 315–328 (1994). https://doi.org/10.1016/0304-3975(92)00011-F

News and Conference Reports



REPORT ON ICALP 2023

50th EATCS International Colloquium on Automata, Languages and Programming

Anca Muscholl[1]

The *50th EATCS International Colloquium on Automata, Languages and Programming (**ICALP 2023**)*, the flagship conference and annual meeting of the *European Association for Theoretical Computer Science (EATCS)*, took place in Paderborn, on July 10-14, 2023. The event was organized by the Computer Science Department of Paderborn University.

The main conference was preceded by a series of workshops on July 10, 2023. The following workshops were held as satellite events of ICALP 2023:

- Combinatorial Reconfiguration
- Graph Width Parameters: from Structure to Algorithms (GWP 2023)
- Algorithmic Aspects of Temporal Graphs VI
- Adjoint Homomorphism Counting Workshop (ad hoc)
- Congestion Games
- Workshop On Reachability, Recurrences, and Loops '23 (WORReLL'23)
- Workshop on Recent Trends in Online Algorithms
- Quantum Computing with Qiskit, and why Classical Algorithms still matter!
- Algebraic Complexity Theory

The scientific programme of ICALP 2023 consisted of 2 unifying lectures and 3 invited lectures of the two tracks of ICALP, the presentation of 132 contributed papers (which were selected by the Program Committees out of 443 submissions) and award sessions with the lectures of the EATCS Award 2023, the Presburger Award 2023 and the Alonzo Church Award 2023 recipients.

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

Organisation of ICALP 2023 and ICALP 50th special event. ICALP 2023 took place at the Heinz-Nixdorf MuseumsForum and the Heinz-Nixdorf Institut in Paderborn, Germany (same venue as for ICALP 1996). The conference was primarily on-site, with 9 exceptions for remote presentations via Zoom (typically due to travel visa delays). Talks were 20 minutes in length, with 5 minutes for questions and for switching rooms between tracks. The conference had a SafeToC representative, Zahra Raissi. Early registration costs were 640 EUR for ICALP and 75 EUR for workshops (440 EUR and 75 EUR for students, respectively), and regular registration was 740 EUR and 100 EUR (540 EUR and 100 EUR, respectively). This was approximately 100 EUR higher than ICALP 2022 in Paris, but included all lunches and most evenings' dining. A somewhat non-conventional measure adopted this year was to require that each accepted paper have at least one *regular* registration (as opposed to *student* registration) — this ensured a lower bound on income through registration numbers, and minimized financial risk during planning. The workshops and conference had 226 and 267 participants, respectively, with a total of 335 unique participants combined for both events. The conference sponsors were DeepL (gold sponsor), SFB 901 On-The-Fly Computing (gold), Reply (silver), Stiebel Eltron (silver), PC^2 Paderborn Center for Parallel Computing (silver), Google Research (bronze), Heinz-Nixdorf Institut (bronze).

Special events. As this was the 50th occurrence of ICALP, various special events were planned. All registrants had free access to the Heinz-Nixdorf MuseumForum's 6000m^2 Computer Science museum, spanning computing technologies from 3000 BC to the present. A special "50th ICALP anniversary session" featured additional invited talks by Kurt Mehlhorn (MPI) and Thomas Henzinger (ISTA). There was also a Colloquium in Honor of Friedhelm Meyer auf der Heide immediately after ICALP, and informally tied to ICALP, featuring invited talks by Artur Czumaj (Warwick), Kurt Mehlhorn (MPI), Christian Sohler (Cologne), and Martin Dietzfelbinger (Ilmenau). Finally, special social events included: (1) A Paderborn city tour and reception at the historical City Hall, (2) an outdoor German BBQ extravaganza, and (3) the conference dinner at the 160-year old Strate Brewery in Detmold, where brewery tours, a German brass band, and unlimited craft beers of various flavors were on tap.

Organizing committee. The organizing committee consisted of Johannes Blömer, Sevag Gharibian (chair), Friedhelm Meyer auf der Heide, Christian Scheideler, and Ulf-Peter Schroeder. A particular thank-you is due to Ulf-Peter, who pulled many of the strings behind the scenes.

On behalf of the entire community, we warmly thank all the organizers in

	2023	2022	2021	2020	2019	2018	2017	2016	2015	2014
Total 443	433	462	470	490	502	459	515	507	477	
Track A	346	350	361	347	316	346	296	319	328	312
Track B	97	83	101	123	103	96	108	121	114	106
Track C	—	—	—	—	71	60	55	75	65	59

Table 1: Number of submitted papers at ICALP 2014–2023.

Paderborn for their fantastic efforts that helped to make ICALP 2023 and the 50th anniversary of ICALP a very successful and pleasant event.

Paper selection and work of the Program Committee. Since 2020 ICALP returned to the original two-track format. The ICALP 2023 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 2023 were Uriel Feige (Track A) and Kousha Etessami (Track B). The two Program Committees involved 65 members (40 in track A and 25 in track B). In response to the call for papers, a total of 443 submissions were received: 346 for Track A and 97 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 132 papers for inclusion in the scientific program: 103 papers for Track A and 29 for Track B. This gives the acceptance rate for the entire conference to be 29.7%. 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 all 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 2023 featured five invited presentations, all available online at `https://videos.uni-paderborn.de/channel/ICALP-2023/94`.

	2023	2022	2021	2020	2019	2018	2017	2016	2015	2014
Total	132	127	137	138	146	147	137	146	143	136
Track A	103	103	108	102	94	98	88	89	89	87
Track B	29	24	29	36	31	30	32	36	34	31
Track C	—	—	—	—	21	19	17	21	20	18

Table 2: Number of accepted papers at ICALP 2014–2023.

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

Table 3: Acceptance rates (in %) for ICALP 2014–2023.

- Anna Karlin (University of Washington, USA), *(Slightly) Improved Approximation Algorithm for the Metric Traveling Salesperson Problem*,
- Rasmus Kyng (ETH Zurich, Switzerland), *An Almost-Linear Time Algorithm for Maximum Flow and More*,
- Rupak Majumdar (Max Planck Institute for Software Systems, Germany), *Context-Bounded Analysis of Concurrent Programs*,
- Thomas Vidick (California Institute of Technology, USA, and Weizmann Institute of Science, Israel), *Quantum codes, local testability and interactive proofs: state of the art and open questions*,
- James Worrell (University of Oxford, UK), *The Skolem Landscape*.

The 50th ICALP anniversary session, chaired by Burkhard Monien (University of Paderborn, Germany), featured two invited talks:

- Kurt Mehlhorn (Max Planck Institute Saarbrücken, Germany), *50 Years of ICALP, Personal Reminiscences*,
- Thomas Henzinger (IST, Austria), *From Formal Methods for Continuous Systems to the Safety of Neural Network Controllers*.

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 132 ICALP 2023 contributed papers presented at the conference, and papers or abstracts accompanying the invited talks, were published according to the principle of Open Access in LIPIcs Proceedings volume 261, and made available online and free of charge at `https://drops.dagstuhl.de/opus/volltexte/2023/18051/`. The proceedings editors are Kousha Etessami, Uriel Feige and Gabriele Puppis.

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 Papers in Track A: Tsun-Ming Cheung, Hamed Hatami, Pooya Hatami, and Kaave Hosseini, *Online Learning and Disambiguations of Partial Concept Classes*, and Miguel Bosch Calvo, Fabrizio Grandoni, and Afrouz Jabal Ameli, *A 4/3 Approximation for 2-Vertex-Connectivity*.

Best Paper in Track B: Marvin Künnemann, Filip Mazowiecki, Lia Schütze, Henry Sinclair-Banks, and Karol Wegrzycki, *Coverability in VASS Revisited: Improving Rackoff's Bound to Obtain Conditional Optimality*.

Best Student Paper in Track A: Manuel Cáceres, *Minimum Chain Cover in Almost Linear Time*.

Best Student Paper in Track B: Ruiwen Dong, *The Identity Problem in $\mathbb{Z} \wr \mathbb{Z}$ is decidable*.

The program of ICALP 2023 included presentations of several prestigious scientific awards sponsored or co-sponsored by EATCS:

- The **EATCS Award 2023**, 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 **Amos Fiat** (Tel Aviv University) for his fundamental work within many areas of theoretical computer science and in particular for work in cryptography, on-line algorithms, and algorithmic game theory.[2]

[2]The laudatio for the EATCS Award 2023 is available at `https://eatcs.org/index.php/component/content/article/1-news/2939-the-eatcs-award-2023-laudatio-for-amos-fiat`.

- The **Presburger Award 2023** for Young Scientists was jointly awarded to **Aaron Bernstein** (Rutgers University) for his breakthroughs in devising fast graph algorithm, and to **Thatchaphol Saranurak** (University of Michigan) for his strong impact in developing new techniques for expander decomposition.[3]

- The **Alonzo Church Award 2023** for Outstanding Contributions to Logic and Computation went to the following group of papers for the design and implementation of Iris, a higher-order concurrent separation logic framework:

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

- The **EATCS Distinguished Dissertation Awards 2023**, to promote and recognize outstanding dissertations in theoretical computer science were awarded to three researchers:

  - **Kuikui Liu** (University of Washington) for his dissertation *Spectral Independence: A New Tool to Analyze Markov Chains*, supervised by Shayan Oveis Gharan.
  - **Alex Lombardi** (MIT, Department of Electrical Engineering and Computer Science) for his dissertation *Provable Instantiations of Correlation Intractability and the Fiat-Shamir Heuristic*, supervised by Vinod Vaikuntanathan,
  - **Lijie Chen** (MIT, Department of Electrical Engineering and Computer Science) for his dissertation *Better Hardness via Algorithms, and New Forms of Hardness versus Randomness*, supervised by Ryan Williams.

[3]The laudatio for the Presburger Award 2023 is available at `https://eatcs.org/index.php/component/content/article/1-news/2950-presburger-award-2023-laudatio`.

- The EATCS has recognized three of its members for their outstanding contributions to theoretical computer science by naming them **EATCS Fellows** class of 2023:
  - Michael A. Bender (Stoney Brook University),
  - Leslie Ann Goldberg (University of Oxford),
  - Claire Mathieu (CNRS, IRIF, Université de Paris).

The recipients of the EATCS Award 2023 (Amos Fiat), of the Presburger Award 2023 (Aaron Bernstein and Thatchaphol Saranurak), and of the Alonzo Church Award 2023 (Ralf Jung, Derek Dreyer and Robbert Krebbers) gave presentations in the Award Sessions.

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 50th ICALP an unforgettable conference. Everyone involved in the organization of ICALP 2023 deserves the warmest thanks from the TCS community.

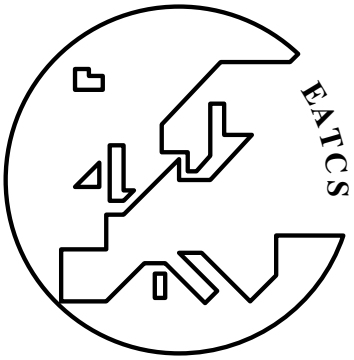
We wish to thank all authors who submitted extended abstracts for consideration, the Program Committees for their effort, and all the referees who assisted the Program Committees in the evaluation process. We are particularly grateful to the Conference Chair Sevag Gharibian for the perfect organisation of the conference. We also acknowledge financial support from DeepL, SFB 901 On-The-Fly Computing, Reply, Stiebel Eltron, PC^2 Paderborn Center for Parallel Computing, Google Research, and the Heinz-Nixdorf Institut.

The General Assembly of the EATCS was informed about the progress of the organization of the *51st EATCS International Colloquium on Automata, Languages and Programming*, ICALP 2024, that will take place in Tallinn, Estonia on July 10–14, 2024, with Pawel Sobocinski as the general chair of the conference. The ICALP 2024 Program Committee Chairs are Ola Svensson and Karl Bringmann (Track A) and Martin Grohe (Track B). ICALP 2024 will be co-located with *IEEE Logic in Computer Science (LiCS)*.

We hope that you will make plans to submit your best work to ICALP 2024 and be able to go to Tallinn for the conference. We look forward to seeing you there.

BEATCS no 141

European Association for Theoretical Computer Science



E A T C S

EATCS

HISTORY AND ORGANIZATION

EATCS is an international organization founded in 1972. Its aim is to facilitate the exchange of ideas and results among theoretical computer scientists as well as to stimulate cooperation between the theoretical and the practical community in computer science.

Its activities are coordinated by the Council of EATCS, which elects a President, Vice Presidents, and a Treasurer. Policy guidelines are determined by the Council and the General Assembly of EATCS. This assembly is scheduled to take place during the annual **I**nternational **C**olloquium on **A**utomata, **L**anguages and **P**rogramming (ICALP), the conference of EATCS.

MAJOR ACTIVITIES OF EATCS

- Organization of ICALP;
- Publication of the "Bulletin of the EATCS;"
- Award of research and academic career prizes, including the EATCS Award, the Gödel Prize (with SIGACT), the Presburger Award, the EATCS Distinguished Dissertation Award, the Nerode Prize (joint with IPEC) and best papers awards at several top conferences;
- Active involvement in publications generally within theoretical computer science.

Other activities of EATCS include the sponsorship or the cooperation in the organization of various more specialized meetings in theoretical computer science. Among such meetings are: CIAC (Conference of Algorithms and Complexity), CiE (Conference of Computer Science Models of Computation in Context), DISC (International Symposium on Distributed Computing), DLT (International Conference on Developments in Language Theory), ESA (European Symposium on Algorithms), ETAPS (The European Joint Conferences on Theory and Practice of Software), LICS (Logic in Computer Science), MFCS (Mathematical Foundations of Computer Science), WADS (Algorithms and Data Structures Symposium), WoLLIC (Workshop on Logic, Language, Information and Computation), WORDS (International Conference on Words).

Benefits offered by EATCS include:

- Subscription to the "Bulletin of the EATCS;"
- Access to the Springer Reading Room;
- Reduced registration fees at various conferences;
- Reciprocity agreements with other organizations;
- 25% discount when purchasing ICALP proceedings;
- 25% discount in purchasing books from "EATCS Monographs" and "EATCS Texts;"
- Discount (about 70%) per individual annual subscription to "Theoretical Computer Science;"
- Discount (about 70%) per individual annual subscription to "Fundamenta Informaticae."

Benefits offered by EATCS to Young Researchers also include:

- Database for Phd/MSc thesis
- Job search/announcements at Young Researchers area

(1) THE ICALP CONFERENCE

ICALP is an international conference covering all aspects of theoretical computer science and now customarily taking place during the second or third week of July. Typical topics discussed during recent ICALP conferences are: computability, automata theory, formal language theory, analysis of algorithms, computational complexity, mathematical aspects of programming language definition, logic and semantics of programming languages, foundations of logic programming, theorem proving, software specification, computational geometry, data types and data structures, theory of data bases and knowledge based systems, data security, cryptography, VLSI structures, parallel and distributed computing, models of concurrency and robotics.

SITES OF ICALP MEETINGS:

- Paris, France 1972
- Saarbrücken, Germany 1974
- Edinburgh, UK 1976
- Turku, Finland 1977
- Udine, Italy 1978
- Graz, Austria 1979
- Noordwijkerhout, The Netherlands 1980
- Haifa, Israel 1981
- Aarhus, Denmark 1982
- Barcelona, Spain 1983
- Antwerp, Belgium 1984
- Nafplion, Greece 1985
- Rennes, France 1986
- Karlsruhe, Germany 1987
- Tampere, Finland 1988
- Stresa, Italy 1989
- Warwick, UK 1990
- Madrid, Spain 1991
- Wien, Austria 1992
- Lund, Sweden 1993
- Jerusalem, Israel 1994
- Szeged, Hungary 1995
- Paderborn, Germany 1996
- Bologne, Italy 1997
- Aalborg, Denmark 1998
- Prague, Czech Republic 1999
- Genève, Switzerland 2000
- Heraklion, Greece 2001
- Malaga, Spain 2002
- Eindhoven, The Netherlands 2003
- Turku, Finland 2004
- Lisabon, Portugal 2005
- Venezia, Italy 2006
- Wrocław, Poland 2007
- Reykjavik, Iceland 2008
- Rhodes, Greece 2009
- Bordeaux, France 2010
- Zürich, Switzerland 2011
- Warwick, UK 2012
- Riga, Latvia 2013
- Copenhagen, Denmark 2014
- Kyoto, Japan 2015
- Rome, Italy 2016
- Warsaw, Poland 2017
- Prague, Czech Republic 2018
- Patras, Greece 2019
- Saarbrücken, Germany (virtual conference) 2020
- Glasgow, UK (virtual conference) 2021
- Paris, France 2022
- Paderborn, Germany 2023

(2) THE BULLETIN OF THE EATCS

Three issues of the Bulletin are published annually, in February, June and October respectively. The Bulletin is a medium for *rapid* publication and wide distribution of material such as:

- EATCS matters;
- Technical contributions;
- Columns;
- Surveys and tutorials;
- Reports on conferences;
- Information about the current ICALP;
- Reports on computer science departments and institutes;
- Open problems and solutions;
- Abstracts of Ph.D. theses;
- Entertainments and pictures related to computer science.

Contributions to any of the above areas are solicited, in electronic form only according to formats, deadlines and submissions procedures illustrated at `http://www.eatcs.org/bulletin`. Questions and proposals can be addressed to the Editor by email at `bulletin@eatcs.org`.

(3) OTHER PUBLICATIONS

EATCS has played a major role in establishing what today are some of the most prestigious publication within theoretical computer science.

These include the *EATCS Texts* and the *EATCS Monographs* published by Springer-Verlag and launched during ICALP in 1984. The Springer series include *monographs* covering all areas of theoretical computer science, and aimed at the research community and graduate students, as well as *texts* intended mostly for the graduate level, where an undergraduate background in computer science is typically assumed.

Updated information about the series can be obtained from the publisher.

The editors of the EATCS Monographs and Texts are now M. Henzinger (Vienna), J. Hromkovič (Zürich), M. Nielsen (Aarhus), G. Rozenberg (Leiden), A. Salomaa (Turku). Potential authors should contact one of the editors.

EATCS members can purchase books from the series with 25% discount. Order should be sent to:

Prof.Dr. G. Rozenberg, LIACS, University of Leiden,
P.O. Box 9512, 2300 RA Leiden, The Netherlands

who acknowledges EATCS membership and forwards the order to Springer-Verlag.

The journal *Theoretical Computer Science*, founded in 1975 on the initiative of EATCS, is published by Elsevier Science Publishers. Its contents are mathematical and abstract in spirit, but it derives its motivation from practical and everyday computation. Its aim is to understand the nature of computation and, as a consequence of this understanding, provide more efficient methodologies.

The Editor-in-Chief of the journal currently are D. Sannella (Edinburgh), L. Kari and P.G. Spirakis (Patras).

ADDITIONAL EATCS INFORMATION

For further information please visit `http://www.eatcs.org`, or contact the President of EATCS:

Prof. Artur Czumaj,
Email: `president@eatcs.org`

EATCS MEMBERSHIP

DUES

The dues are € 40 for a period of one year (two years for students / Young Researchers). Young Researchers, after paying, have to contact `secretary@eatcs.org`, in order to get additional years. A new membership starts upon registration of the payment. Memberships can always be prolonged for one or more years.

In order to encourage double registration, we are offering a discount for SIGACT members, who can join EATCS for € 35 per year. We also offer a five-euro discount on the EATCS membership fee to those who register both to the EATCS and to one of its chapters. Additional € 35 fee is required for ensuring the *air mail* delivery of the EATCS Bulletin outside Europe.

HOW TO JOIN EATCS

You are strongly encouraged to join (or prolong your membership) directly from the EATCS website `www.eatcs.org`, where you will find an online registration form and the possibility of secure online payment. Alternatively, contact the Secretary Office of EATCS:

Mrs. Efi Chita,
Computer Technology Institute & Press (CTI)
1 N. Kazantzaki Str., University of Patras campus,
26504, Rio, Greece
Email: `secretary@eatcs.org`,
Tel: +30 2610 960333, Fax: +30 2610 960490

If you are an EATCS member and you wish to prolong your membership or renew the subscription you have to use the Renew Subscription form. The dues can be paid via paypal and all major credit cards are accepted.

For adittional information please contact the Secretary of EATCS:

Dmitry Chistikov
Computer Science
University of Warwick
Coventry
CV4 7AL
United Kingdom
Email: `secretary@eatcs.org`,