COUNCIL OF THE
EUROPEAN ASSOCIATION FOR
THEORETICAL COMPUTER SCIENCE

President: Giorgio Ausiello
Vice Presidents: Burkhard Monien, Don Sannella, Paul Spirakis
Treasurer: Dirk Janssens
Bulletin Editor: Maria Serna

Luca Aceto (Iceland) Madhavan Mukund (India)
Krzysztof Apt (The Netherlands) Mogens Nielsen (Denmark)
Pierre-Louis Curien (France) Catuscia Palamidessi (France)
Josep Diaz (Spain) David Peleg (Israel)
Zoltán Ésik (Hungary) Vladimir Sassone (United Kingdom)
Fedor Fomin (Norway) Jiří Sgall (Czech Republic)
Giuseppe F. Italiano (Italy) Andrzej Tarlecki (Poland)
Juhani Karihumäki (Finland) Wolfgang Thomas (Germany)
Richard E. Ladner (USA) Ingo Wegener (Germany)
Jan van Leeuwen (The Netherlands) Emo Welzl (Switzerland)
Eugenio Moggi (Italy) Gerhard Woeginger (The Netherlands)

Past Presidents:
EATCS COUNCIL MEMBERS

EMAIL ADDRESSES

Luca Aceto ........................................ luca@ru.is
Giorgio Ausiello ................................. ausiello@dis.uniroma1.it
Krzysztof Apt .............................. apt@cwi.nl
Pierre-Louis Curien .......................... curien@pps.jussieu.fr
Josep Díaz ...................................... diaz@lsi.upc.es
Zoltán Ésik ................................. ze@inf.u-szeged.hu
Fedor Fomin .................................. fomin@ii.uib.no
Giuseppe F. Italiano ......................... italiano@disp.uniroma2.it
Dirk Janssens ............................... Dirk.Janssens@ua.ac.be
Juhani Karhumäki .......................... karhumak@cs.utu.fi
Richard E. Ladner ........................... ladner@cs.washington.edu
Jan van Leeuwen .............................. jan@cs.uu.nl
Eugenio Moggi ................................ moggi@disi.unige.it
Burkhard Monien ............................. bm@upb.de
Madhavan Mukund ......................... madhavan@cmi.ac.in
Mogens Nielsen ............................. mn@brics.dk
Catuscia Palamidessi ....................... catuscia@lix.polytechnique.fr
David Peleg .................................. peleg@wisdom.weizmann.ac.il
Don Sannella ................................. dts@dcs.ed.ac.uk
Vladimiro Sassone .......................... vs@ecs.soton.ac.uk
Jiří Sgall ........................................... sgall@math.cas.cz
Maria Serna ................................... mjserna@lsi.upc.edu
Paul Spirakis ................................. spirakis@cti.gr
Andrzej Tarlecki ............................ tarlecki@mimuw.edu.pl
Wolfgang Thomas ......................... thomas@informatik.rwth-aachen.de
Ingo Wegener .......................... ingo.wegener@uni-dortmund.de
Emo Welzl ....................................... emo@inf.ethz.ch
Gerhard Wöeginger ....................... g.j.woeginger@math.utwente.nl
Bulletin Editor: Maria Serna, Barcelona, Spain
Cartoons: DADARA, Amsterdam, The Netherlands

The bulletin is entirely typeset by \TeX{} and \LaTeX{} in \TeX{}-fonts. The Editor is grateful to Uffe H. Engberg, Hans Hagen, Marloes van der Nat, and Grzegorz Rozenberg for their support.

All contributions are to be sent electronically to bulletin@eatcs.org

and must be prepared in \LaTeX{} using the class beatcs.cls (a version of the standard \LaTeX{} 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 laid 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.

The EATCS home page is http://www.eatcs.org
# Table of Contents

## EATCS Matters
- **Letter from the President** .................................................. 3
- **Letter from the Editor** .................................................. 6
- **The EATCS Award 2009** .................................................. 7
- **Ingo Wegener (1950–2008)** ........................................... 10

## EATCS News
- **The Japanese Chapter, by K. Makino** .................................... 17
- **News from India, by M. Mukund** ........................................ 21
- **News from Ireland, by A.K. Seda** ....................................... 24
- **News from Latin America, by A. Viola** .................................. 26
- **News from New Zealand, by C.S. Calude** ................................ 29

## The EATCS Columns
- **The Algorithmic Game Theory Column, by M. Mavronikolas**
  - **Mechanism Design for Scheduling,** by G. Christodoulou and
    E. Koutsoupias .......................................................... 39
- **The Complexity Column, by J. Torán**
  - **The Complexity of Planar Graph Isomorphism,** by J. Torán and
    F. Wagner ................................................................. 60
- **The Distributed Computing Column, by P. Fotouhou**
  - **The Theory of Transactional Memory,** by R. Guerrou and
    M. Kapadka ................................................................. 83
- **The Formal Specification Column, by H. Ehrig**
  - **Graph Abstraction for a Modal Logic,** by I. Boneva, J. Kreiker,
    M. Kurbán and A. Rensik .............................................. 106
  - **Behaviour Simulation and Equivalence of Systems Modelled**
    by Graph Transformation,** by H. Ehrig, C. Ermei, F. Hermann
    and B. König ............................................................ 117
- **The Logics in Computer Science Column, by Y. Gurevich**
  - **Fixed Point Theorem and Aperiodic Tilings,** by B. Durand, A.
    Romashchenko and A. Shen ............................................ 126
- **The Natural Computing Column, by G. Rozenberg**
  - **Physical Experiments as Oracles,** by E. Beggs J.F Costa and
    V. Tucker ................................................................. 137
THE PUZZLE CORNER, by L. Rosaz ................................. 153

MISCELLANEOUS
EVOLUTION OF MUTATING SOFTWARE by G.J. Chaitin .................. 157

REPORTS FROM CONFERENCES
AUTOMATA, FORMAL LANGUAGES AND ALGEBRAIC SYSTEMS (AHLAS 2008), by M. Kudlek .............................................. 167
INTERNATIONAL CONFERENCE ON CONCURRENCY, SPECIFICATION AND PROGRAMMING (CS&P 2008), by M. Kudlek ..................... 169
12TH INTERNATIONAL CONFERENCE ON DEVELOPMENT OF LANGUAGE THEORY (DLT 2008), by M. Kudlek ............................ 171
INTERNATIONAL WORKSHOP ON THE COMPLEXITY OF SIMPLE PROBLEMS, by H. Zenil .................................................. 175
WORKSHOP ON GRAND CHALLENGES OF UNCONVENTIONAL COMPUTATION, by I. Potapov ........................................... 178

ABSTRACTS OF PHD THESES ........................................ 181
PICTURES FROM CONFERENCES .................................... 187
EATCS LEAFLET ....................................................... 211
EATCS MATTERS
Dear EATCS members,

As it is usual in this time of the year I have the pleasure to announce, first of all, the assignment of EATCS Award: the EATCS Award 2009 has been assigned to Gérard Huet for his outstanding contributions to theoretical computer science. The nomination letter, published in this issue of the Bulletin, illustrates the main steps of his outstanding scientific career, ranging from his first contribution on unification of typed lambda terms to the design of the CAML functional programming language, up to the implementation of the Coq proof assistant, today one of the most used and trusted platforms for formalized mathematics and formal methods. The proposal has been made by the selection Committee, chaired by Catuscia Palamidessi and consisting of Paul Spirakis and Emo Welzl, and it has been unanimously approved by the EATCS Council members. The Award will be presented in a ceremony that will take place during ICALP in Rhodes. Let us convey to Gérard the congratulations of the whole EATCS community.

The organization of ICALP 2009 is proceeding smoothly. The submission deadline is now close. The Program Committees, respectively chaired by Susan Albers (Track A), Wolfgang Thomas (Track B) and Alberto Marchetti-Spaccamela and Yossi Matias (co-Chairs of Track C), will have, as usual, to work hard to select the best papers for the three tracks. Remember that while Track A and Track B will be as usual devoted to Algorithms and Complexity, and Logic and Semantics respectively, this year
Track C will be devoted to a new domain "Foundations of Networked Computation: Models, Algorithms and Information Management". The program will be known after the notification date, April 6. The Conference will also host four satellite workshops: ALGOSENSORS 2009 (5th International Workshop on Algorithmic Aspects of Wireless Sensor Networks), DCM 2009 (5th International Workshop on Developments in Computational Models), FOCLASA 2009 (8th International Workshop on Foundations of Coordination Languages and Software Architectures), QUANTLOG 2009 (Workshop on Quantitative Logics). Also the strong names of the invited speakers will make this conference really outstanding. The General Chairs Paul Spirakis, Elias Koutsoupias, Christos Kaklamanis together with their teams, are doing and excellent job. For more information please look at ICALP 2009 website: http://icalp09.cti.gr/.

Also the organization of ICALP 2010 has already started. As you know next year ICALP will be organized in Bordeaux by Claude Kirchner and Cyril Gavoille. The conference will host again three tracks, as in Rhodes this year. The Program Committees of the traditional Tracks A and B will be chaired by Paul Spirakis and Samson Abramsky respectively. Track C will again be devoted to Foundations of Networked Computation and will be chaired by Friedhelm Meyer auf der Heide.

Let me conclude this letter with a sad note. On November 26, 2008, after a three-year struggle with a terrible illness, our friend and colleague Ingo Wegener, one of the most brilliant figures in theoretical computer science in Europe...
and worldwide, has left this world. You can find the obituary in this bulletin issue. On behalf of EATCS I wish to express his wife Christa and all his colleagues the deepest mourning of our community.
Sincere greetings to all of you.

Giorgio Ausiello, Rome
February 2009
Letter from the Bulletin Editor

Dear Reader,

First of all, I wish all of you a happy and successful 2009. Let it be a fruitful year for each of you, for theoretical computer science, and for EATCS.

The major novelty in this issue is the first contribution of the new column: THE ALGORITHMIC GAME THEORY COLUMN edited by Marios Mavronikolas. I hope that you will enjoy the interesting initial contribution to this topic: "Mechanism design for scheduling" by George Christodoulou and Elias Koutsoupias. I have also more good news to report. Starting from this issue, the Bulletin has another new columnist, as to Panagiota Fatourou takes over Marios Mavronikolas as the new editor of the Column on Distributed Computing. Panagiota’s research deals mainly with design and analysis of algorithms (with emphasis to distributed algorithms), distributed computing, and parallel computing. Panagiota is an assistant professor at the Department of Computer Science of the University of Ioannina, Greece, actually on leave and under appointment at the University of Crete.

I trust that you will find a lot of interesting material to read in this issue that will help to keep you up to date with the research and activities in theoretical computer science all over the world. I want to express my gratitude to all the contributors and encourage you to do so in the near future.

Maria Serna, Barcelona
February 2009
The EATCS Award 2009

Nomination of Gérard Huet

The EATCS Award is awarded annually in recognition of a distinguished career in theoretical computer science. The Committee, consisting of Catuscia Palamidessi (Chair), Paul Spirakis and Emo Welzl in charge of evaluating the nominations to the 2009 EATCS Award has come to the decision to propose

Professor Gérard Huet

as the candidate for the 2009 EATCS Award in view of the excellent research contributions to theoretical computer science produced throughout his outstanding scientific career. The Committee unanimously shares the motivations contained in the nomination letter. The proposal has been unanimously approved by the EATCS Council. The Award will be assigned during a ceremony that will take place in Rhodes (Greece) during ICALP2009 (July 5-12, 2009).

Nomination of Prof. Gérard Huet for the EATCS Award 2009

Gérard Huet has made numerous, enduring advances in the foundations of computer science and has been a central figure in several important software systems. He has also had a remarkably active and successful academic and professional life. His distinguished career in theoretical computer science has exerted considerable influence on not only the field but also the many students and colleagues that he has directly influenced.

A hallmark of Huet research is his talent for taking highly technical material and providing it with a clear and deep analysis. For example, in his paper on the unification of typed lambda-terms (in the first volume of TCS, 1975), Huet took a difficult topic, reshaped and redefined it, and left a solution so well developed that it took more than 15 years of active research before any one saw a need to extend it. Since that first major result, he has repeated this performance numerous times.

Equally characteristic of Huet’s research is the intimate connections he maintains between theoretical computer science topics and their effective implementation. He was one of the first computer scientists who was able to move between...
these two domains and who felt that there was no option to doing so: these two topics were absolutely needed to inform each other.

Huet was a leader in the general areas of logical frameworks and constructive typed theories. Thanks in large part to his achievements and efforts, formal mathematics has taken huge strides and is starting to have an impact on the wider world. He has worked extensively at disseminating his view of this bold new world of formalized reasoning: for example, he has organized numerous summer schools and given countless invited talks and tutorials. Many researchers in France and elsewhere count themselves as students of Huet even if they were never formally his PhD advisee.

**Principal Scientific Contributions**

We list and briefly describe some of the many contributions made by Gérard Huet.

- In the early 70’s, Huet developed both resolution and unification for higher-order logic: these results have became the core of several modern systems that perform deduction in higher-order logic.

- Huet has done fundamental research in the areas of rewriting and Knuth-Bendix completion. His writings in this area are extensive and elegant.

- Between 1982 and 1989, Huet directed and contributed to the design and implementation of the CAML functional programming language. That language and its descendants have given academics and industries an efficient and well structured programming language.

- In the 1990s, Huet and his students designed and built the first version of the Coq proof assistant. Today, Coq is one of the most used and trusted platforms for formalized mathematics and formal methods.

Huet has a broad culture inside and outside of computer science. He has made, for example, important contributions in other areas as well: he is the author of executable lecture notes; he is the designer of the Zipper data structure; and he has built the Zen toolbox for phonological and morphological segmentation and labeling of Sanskrit.

**Brief Biography**

In 1972, Huet received his PhD in Computer Science from Case Western Reserve University, Ohio, USA and in the same year, he started as a researcher at IRIA (which later became INRIA), a position that he has maintained since then. In 1976 he obtained his Thèse d’Etat from University Paris 7. In addition to being a
researcher at INRIA, he has been a Professor at the University of Orsay in 1975-76; an Invited Professor at Carnegie Mellon University during 1985-1986; and a Visiting Professor at the Asian Institute of Technology in Bangkok in 1986.

He organized the Institute of Logical Foundations of Functional Programming during the Year of Programming at the University of Texas in Austin in Spring 1987. He organized CEFIPRA workshops at IIT Kanpur in 1990 and 1995, and a series of Winter schools at Institute of Physics in Bhubaneshwar (Orissa) in 1999, 2000, and 2002. He has taught at a number of summer schools, including the Marktoberdorf Summer School, the Toulouse Summer School on Logic and Informatics, the Ecole du Greco de Programmation, and the ESSLLI Summer School. He was the conference organizer for the 5th International Conference on Automated Deduction (CADE) in Les Arcs in 1980 and for the Logic in Computer Science Symposium (LICS) in Paris in 1994.

He has edited three books and authored more than 80 research papers. He has also directed 19 doctoral thesis (including T. Coquand, G. Dowek, C. Paulin-Mohring, and X. Leroy) and took part to more than 100 thesis committees internationally. Huet was also the coordinator of the ESPRIT European projects Logical Frameworks and TYPES from 1990 to 1995.

Huet has received a number of honors and awards to date. He has been Directeur de Recherches de Classe Exceptionnelle at INRIA since 1989. He was elected a Member of Academia Europaea in 1989 and a Member of the Academy of Sciences of the Institute of France in 2002. He received the Herbrand Award for his work on Computational Logic in 1998 and received the degree of Doctor of Technology honoris causa from Chalmers University in Göteborg (Sweden) in 2004.

signed by:

Henk Barendregt Robert Constable
Gilles Dowek
Masami Hagiya
Jean-Pierre Jouannaud
Jean-Jacques Levy
Dale Miller
Robin Milner
Tobias Nipkow
Larry Paulson
Frank Pfenning
Benjamin Pierce
IN MEMORIAM

PROF. DR. INGO WEGENER
(1950–2008)

Ingo Wegener passed away on November 26, 2008, after a three-year struggle with brain cancer. A farewell celebration was held in Bielefeld on December 4, the day on which Ingo would have turned 58. This is a tremendous loss for all those who knew him personally, for Computer Science in Germany, in Europe, and in the world at large, and for many other communities in which he played a prominent role. Ingo is survived by his wife, Christa.

Ingo studied mathematics at the University of Bielefeld, where he graduated with a Diplom in 1976, received his PhD in 1978, with a thesis on circuit complexity, and obtained his Habilitation in 1981. He held a professorship in computer science in Frankfurt am Main from 1980 to 1987, when he became a full professor of Computer Science (in the area of Efficient Algorithms and Complexity Theory) at the Technische Universität Dortmund, the position he held until his death.

Ingo had an amazing talent for identifying interesting research areas and for asking exactly the right questions. More often than not, he also came up with the answers himself, by providing novel and efficient algorithms and proof techniques. His versatility and comprehensive knowledge in many different areas was astounding. Many younger researchers will know Ingo from his work in the years starting from the early 1990s, when he started to work in a newly created research area which might be called Formal analysis of metaheuristics. It was his conviction that the huge field of optimization algorithms based on metaheuristics like
Evolutionary Algorithms, Simulated Annealing, and so on, should be studied using the methods from the theory of efficient algorithms and complexity theory. Previously, the standard approach to measuring the efficiency of the heuristics had been to simply implement them and run tests. This new, theoretical, approach turned out to be very fruitful, yielding a much deeper understanding of the limitations of such metaheuristics. It was documented in a large number of scientific papers by Ingo, by members of his research group, by his scientific cooperators in Germany and abroad, and many others.

Equally important was the work of Ingo and his group in the area of algorithms for and complexity of branching programs and Binary Decision Diagrams. Ingo thoroughly studied this fundamental method for representing Boolean functions, interesting as an object of theoretical study and eminently useful in practice. He posed many important and ultimately fruitful questions, and introduced several important variants of the basic OBDD model. He furthered the accessibility of this area to researchers all over the world by writing the monograph *Branching Programs and Binary Decision Diagrams – Theory and Applications*, guiding the reader to the edge of current research. Before coming to Dortmund in 1987, Ingo had already finished his first influential monograph, *The Complexity of Boolean Functions* – a must on the bookshelf of every complexity theorist of that time. This book was based on his experience in the area of circuit and branching program complexity, to which he contributed several important results. Before that period, he did successful research in the area of search problems, a topic that he revisited in his later work on search heuristics. How he, in addition to these achievements, found the time to do research on algorithms and data structures, like bottom-up heap sort, or knight’s tours on chessboards, is anyone’s guess.

Besides and beyond his scientific achievements, Ingo was a gifted and devoted teacher who managed to fascinate his students in an astounding way. His lectures covered all aspects of Efficient Algorithms and Complexity Theory, ranging from second-year algorithms classes over *Formal Languages, Computability, and Complexity* through advanced graduate courses to very specialized research-related topics. Ingo’s lectures were extremely well prepared, many of them accompanied by lecture notes – not written by scribes on-the-fly, but by himself before the semester started. When he delivered his lectures, complicated thoughts came across very light-handedly. He made them very easy to grasp, never frustratingly difficult. Even his very tough mandatory classes were popular among the students. Ingo was the only professor at TU Dortmund who won the University Price for Excellence in Teaching twice. (The students nominate the prize winner.) He wrote several attractive textbooks for undergraduate students, and he was a devoted advisor for the students who wrote their diploma theses under his guidance (there were more than 130 of them). His enthusiastic manner also made him an ideal mentor for his many PhD students.
For his scientific achievements Ingo was awarded the Konrad-Zuse-Medaille in 2006, the most prestigious German computer science award. He was a member of some of the most important Academies in Germany, notably the German Academy of Sciences Leopoldina. In 2004 he was appointed to the German Council of Science and Humanities (the most important scientific advisory committee to the German government), a great honor and also a source for a lot of work that he applied himself to with great enthusiasm. His advice was valued both within his university – he was a long time member of the academic senate – and in the several steering committees of important German Computer Science endeavors, like the scientific directorate of Schloss Dagstuhl, in which he participated. On a European level, starting 2005, he was a member of the Council of the EATCS.

The basis of all these formally acknowledgeable and countable achievements was the fact that Ingo was simply as devoted a scientist as he was a teacher. He could not help but search for new questions to answer. Dealing with him was pleasurable, because of his calm and pragmatic way of approaching issues, and because of his open and friendly manner. Ingo will be missed by many – students, colleagues, and friends – and for quite a few students and colleagues he was a friend and an important source of thoughtful advice over the years. But he will also live on – in the results of his work, but also in the minds of all those who had the fortune to work with him.

Martin Dietzfelbinger
Institutional Sponsors
BRICS, Basic Research in Computer Science  
Aarhus, Denmark

Microsoft Research  
Cambridge, United Kingdom

Elsevier  
Amsterdam, The Netherlands

Springer-Verlag  
Heidelberg, Germany

CTI, Computer Technology Institute  
Greece
EATCS News
REPORT FROM THE JAPANESE CHAPTER

K. Makino (Tokyo Univ.)

EATCS-JP/LA Workshop on TCS
As announced in the previous report, the Seventh EATCS-JP/LA Workshop on Theoretical Computer Science will be held at Research Institute of Mathematical Sciences, Kyoto Univ., Feb. 2 – 4, 2009. The workshop will be jointly organized with LA, Japanese association of theoretical computer scientists. Its purpose is to give a place for discussing topics on all aspects of theoretical computer science. Please check the program at the end of this report.

On TCS Related Activities in Japan:
TGCOMP Meetings, September – December, 2008
The IEICE, Institute for Electronics, Information and Communication Engineers of Japan, has a technical committee called TGCOMP, Technical Group on foundation of COMPuting. During September – December of 2008, TGCOMP organized 3 meetings and about 30 papers (including three tutorials) were presented there. Topics presented are, very roughly, classified as follows.

- Algorithm: On Graphs (4)
- Algorithm: On Strings (1)
- Algorithm: On Other Objects (7)
- Online Algorithm (2)
- Combinatorics / Probabilistic Analysis (4)
- Computational Complexity and
- Cryptography (5)
- Computational Learning / Knowledge
- Discovery (1)
- Distributed Computing (4)
- Formal Languages and Automata (1)
- Quantum Computing (1)

See our web page for the list of presented papers (title, authors, key words, email).

EATCS-JP/LA Workshop on TCS Program
NOTICE: EATCS-JP/LA Workshop is an unrefereed meeting; that is, all submissions are accepted for the presentation. Thus there should be no problem of presenting these papers in refereed conferences and/or journals.

Mon, Feb. 2
Session 1 (13:00-14:20) (Kyoto U.), Shuji Okamura (Osaka U.)
1. An Probabilistic Algorithm Which Generates All Linear Extensions of a d-Complete Poset, Kento Nakada
2. Collisions in Balls-and-Bins, Toshio Nakata (Fukuoka U. of Education)
3. Privacy-Preserving Datamining Based on Linear Transformation and Random Matrix Theory, Chunhua Su, Kouichi Sakurai (Kyushu U.)

4. Evolutionary Dynamics on Graph Using Random Walk Theory, Yizhi Ren (Kyushu U., Dalian U. of Technology), Mingchu Li (Dalian U. of Technology), Kouichi Sakurai (Kyushu U., ISIT)

Session 2 (14:40-16:00)

5. A Note on Characterizations of Context-Free Languages Using Insertion and Locality, Kaoru Onodera (Tokyo Denki U.)


7. Modeling, Specification and Verification of Embedded systems based on Probabilistic Timed Game Theory, Masashi Hayashi, Satoshi Yamane (Kanazawa U.)

8. Probabilistic Timed CEGAR, Atsushi Morishita, Satoshi Yamane (Kanazawa U.)

Session 3 (16:20-17:40)

9. Enumerating Polyominoes of p4 Tiling by the Reverse Search, Takashi Horiyama, Masato Samejima (Saitama U.)

10. How to Make a Picturesque Maze, Yoshio Okamoto (Tokyo Inst. of Technology), Ryuhei Uehara (JAIST)

11. Complexity of Pleats Foldings, Tsuyoshi Ito (McGill U.), Masashi Kiyomi, Ryuhei Uehara (JAIST)

12. The Hospitals/Residents Problem with Quota Lower Bounds, Kouki Hamada, Shuichi Miyazaki, Kazuo Iwama (Kyoto U.)

Tues, Feb. 3

Session 4 (9:00-10:20)


14. DNA Model Associate to Chomsky Language, Osamu Suzuki (Nihon U.)

15. Formal Language Associate to Knots, Yuichi Uetani, Yosiko Okada, Syun Horiguti, Tatuya Maekawa, Osamu Suzuki (Nihon U.)


Session 5 (10:40-12:00)

17. Chaitin’s Halting Probability Ω and Halting Problems, Kohtaro Tadaki (Chuo U.)

The Bulletin of the EATCS

19. An Improvement of the Soundness of a 3-Bit PCP, Naoki Kinoshita, Suguru Tamaki, Kazuo Iwama (Kyoto U.)

20. Online Learning Based on Maximum Entropy Principle, Takafumi Ohta, Kohei Hatano, Masayuki Takeda (Kyoto U.)

Session 6 (13:30-14:50)

21. The Carving-Width of Cartesian Powers of Regular Graphs, Kyohei KOZAWA, Yoya OTACHI, Koichi YAMAZAKI (Gunma U.)

22. Weighted Alliances in Graphs, Kenji Kimura (The U. of Electro-Communications), Masayuki Koyama, Akira Saito (Nihon U.)

23. On Approximating a Minimum Directed Tree Cover in a Layered Graph, Tetsuma Tada, Toshihiro Fujito (Toyohashi U. of Technology)

24. A Tight Upper Bound on the Cover Time for Metropolis Walks, Yoshiaki Nomaka, Hirotaka Ono, Kumihiro Sadakane, Masafumi Yamashita (Kyushu U.)

Student Session 1 (15:10-16:10)

25. A Simple Characterization of Serially Constructible Episodes, Takashi Katoh, Hiroki Arimura (Hokkaido U.), Koichi Hirata (Kyushu Inst. of Technology)


27. Linear Time Algorithm with O(nloglogn) Bit Space for Ranking of Permutations, Fumiya Suto, Ayumi Shinohara (Tohoku U.)

28. Average Analysis of Repetitions in a String, Kazuhiro Kusano, Wataru Matsubara, Akira Ishino, Ayumi Shinohara (Tohoku U.)

Student Session 2 (16:30-17:30)

29. Another Reduction from Graph Isomorphism to Ring Isomorphism, Tomoyuki Hayasaka (Tokyo Inst. of Technology)


31. Faster Random Walks on Series-Parallel Biconnected Graphs, Yasuyuki YAMASAKI, Eiji MIYANO (Kyushu Inst. of Technology)

32. Approximation of the Minimum Manhattan Network Problem, Yasuyuki YAMASAKI, Eiji MIYANO (Kyushu Inst. of Technology)

Wed, Feb. 4

Session 7 (9:00-10:20)

33. A New Cryptographic System Based on the Difficulty of Automata Identification Problem, Seiya OKUBO (U. of Shizuoka), Tetsuro NISHINO , Mitsuwa WAKATSUKI (The U. of Electro-Communications)
34. **NFALSE: Another Ring-Based Public Key Cryptosystem with Faster Encryption**, Keita Xagawa, Keisuke Tanaka (Tokyo Inst. of Technology)

35. **On Encryption Functions and their Properties**, Takato Hirano, Keisuke Tanaka (Tokyo Inst. of Technology)

36. **A Random Oracle Model with Setting and Watching Queries**, Mario Larangeira, Akira Numayama, Keisuke Tanaka (Tokyo Inst. of Technology)

**Session 8 (10:40-12:00)**

37. **On Existence and Uniqueness of Distance k-Sector and Zone Diagram of Objects**, Keiko Imai (Chuo U.), Akitoshi Kawamura (Tokyo U.), Yu Muramatsu (Chuo U.), Takeshi Tokuyama (Tohoku U.)

38. **Square and Rectangle Covering with Outliers**, Hee-Kap Ahn (POSTECH), Sang Won Bae (KAIST), Sang-Sub Kim (POSTECH), Matias Korman (Tohoku U.), Iris Reinbacher (KAIST)

39. **An Efficient Algorithm for the Nearest Larger Neighbors Problem on Matrices**, Keita Nogi, Tetsuo Asano, Masashi Kiyomi (JAIST)

40. **An Exact Algorithm Computing the Longest Path Length Distribution on DAG with Exponentially Distributed Edge Weights**, Ei Ando, Hirotaka Ono, Kunihiko Sadakane, Masafumi Yamashita (Kyushu U.)

---

**THE JAPANESE CHAPTER**

**Chair:** Kazuo Iwama  
**V. Chair:** Osamu Watanabe  
**Secretary:** Kazuhisa Makino  
**Email:** eatcs-jp@is.titech.ac.jp  
**URL:** http://www.misojiro.t.u-tokyo.ac.jp/eatcs-j
In this edition of News from India, we report on some schools and workshops from this winter.

**FSTTCS/APLAS 2008**

The 28th edition of FSTTCS, the annual conference of the Indian Association for Research in Computing Science (IARCS) was held at the Indian Institute of Science (IISc), Bangalore in December, 2008. The conference formed part of the official centenary celebrations of IISc. The Programme Committee was chaired by Ramesh Hariharan (CMI / Strand Life Sciences), Madhavan Mukund (CMI) and V Vinay (CMI / Geodesic).

FSTTCS 2008 was colocated with the Sixth Asian Symposium on Programming Languages and Systems (APLAS 2008), which was held in India for the first time. The Programme Committee chair for APLAS 2008 was G. Ramalingam (MSR India).

The two conferences ran side-by-side over three days, with a common registration and three technical sessions (two in FSTTCS, one in APLAS) running in parallel at the same venue. Several participants exploited the flexibility to attend sessions across both conferences.

FSTTCS and APLAS had one joint invited speaker, Simon Peyton-Jones (Microsoft Research, Cambridge). In addition, FSTTCS had four other invited talks, by Hubert Comon-Lundh (RCIS, AIST, Tokyo and ENS Cachan), Uriel Feige (Weizmann), Erich Grädel (Aachen) and Leslie Valiant (Harvard), while APLAS had two additional invited talks, by Dino Distefano (Queen Mary, University of London) and Radha Jagadeesan (DePaul University).
The proceedings of FSTTCS 2008 were published in the form of an open electronic archive, hosted on the Dagstuhl Research Online Publication Server (DROPS), with privately printed copies distributed to the participants. This initiative, following a similar move by STACS in 2008, has been well received and will be maintained in future editions of the conference.

The official website of FSTTCS, http://www.fsttcs.org, will soon be updated with full details about invited speakers, accepted papers and other data about all editions of the conference, since its inception in 1981.

TECS Week 2009

The 7th TCS Excellence in Computer Science Week (TECS Week 2009) was held at the Tata Research Development and Design Centre (TRDDC), Pune from 5 to 9 January 2009. It was conducted by TRDDC jointly with United Nations University (UNU/IIST) and the Indian Association for Research in Computing Science (IARCS).

The topic for this year’s TECS Week was Decentralized Cooperative Computing. The invited speakers were Lorenzo Alvisi (Univ Texas at Austin, USA), Peter Druschel (MPI, Germany), S. Keshav (Waterloo, Canada), Ravi Kumar (Yahoo! Research, USA) and Guru Parulkar (Stanford, USA).

Lorenzo Alvisi spoke about Byzantine fault tolerant systems. Peter Druschel brought out technical challenges such as self-organization, robustness, and incentive-compatibility in designing decentralized systems. S. Keshav and Ravi Kumar discussed applications of decentralized cooperative computing. Guru Parulkar addressed the issues arising from the fact that new inventions are pushing the Internet into realms that the original design had not anticipated.

TECS Week 2009 had 65 participants from academic institutions, government research labs, and industry from India and neighboring countries in Asia.

ICLA 2009

The 3rd Indian Conference on Logic and its Applications (ICLA 2009), a biennial conference, was held this year at The Institute of Mathematical Sciences, Chennai, India, on January 9–11, 2009. There were two (parallel) pre-conference workshops on January 7–8, 2009. One was on Algebraic Logic, and the other was on Logic and Social Interaction. The program chairs were R Ramanujam from The Institute of Mathematical Sciences, Chennai, and Sundar Sarukkai from the National Institute for Advanced Studies, Bangalore.

The conference and workshops had about 100 participants from all over India and abroad. The workshop on Algebraic Logic featured tutorial lectures on topics
like relation algebras and games, substructural logics, and residuated lattices, by Robin Hirsch, Ian Hodkinson, Hiroakira Ono, and Peter Jipsen. The workshop on Logic and Social Interaction had invited lectures on topics such as the Logic of Campaigning, the logic of intelligent interaction, multi-agent preference reasoning, by Johan van Benthem, Rohit Parikh, Francesca Rossi, Alexandru Baltag, Jan van Eijck, and Rineke Verbrugge.

The conference had invited talks by Rohit Parikh on Knowledge and games, Rajeev Gore on machine checking proof theory, Joel Hamkins on second order set theory, Moshe Vardi on philosophical logics and industrial logics, Esko Turunen on fuzzy logics, Johan van Benthem on a logical perspective on decisions, actions, and games, and Johann Makowsky on MSO-definable graph parameters. Besides, there were 18 contributed talks on a wide range of topics including logic and games, set theory, epistemic logic, and Buddhist and Nyaya logics.

Acknowledgments Vaishali Sadaphal from TRDDC provided the report on the TECS Week 2009. The report on ICLA 2009 was written by S.P. Suresh from CMI.

Madhavan Mukund, Chennai Mathematical Institute
Secretary, IARCS (Indian Association for Research in Computing Science)
http://www.cmi.ac.in/~madhavan
A very successful international workshop, the International Workshop on the Complexity of Simple Programs (CSP’08), was held in the National University of Ireland, Cork (NUI, Cork) on December 6th and 7th, 2008. Its focus was on models of computation, computational complexity, and decidability. The following areas were of particular interest: universality in simple models of computation and the computational complexity of problems associated with such models; reductions between properties of simple models and well-known unsolved problems (3n+1, etc.); and decidability results that provide lower bounds for the simplest possible computationally universal models. Results on these topics have already been found for a number of models such as Turing machines, Post systems, Tag systems, cellular automata, neural networks, grammars and dynamical systems, and the workshop continued the work in these areas.

All speakers were personally invited and were as follows: Olivier Bournez; Cristian S. Calude; Matteo Cavaliere; Matthew Cook; Liesbeth De Mol; Jérôme Durand-Lose; Manfred Kudlek; Martin Kutrib; Grégory Lafitte; Leonid Levin; Jack Lutz; Fred Lunnon; Maurice Margenstern; Philippe Moser; Niall Murphy; Nicolas Ollinger; Alexander Okhotin; Matt Patitz; Ivan Rapaport; Gaétan Richard; Yurii Rogozhin; Scott Summers; Klaus Sutner; and Sergey Verlan. Their papers were compiled into a local pre-proceedings published by Cork University Press, and an issue of Theoretical Computer Science will be devoted to publishing full-length versions of the papers.

The programme committee consisted of Turlough Neary, Anthony Seda, and Damien Woods. The local organizing committee consisted of the aforementioned
The Bulletin of the EATCS

together with Niall Murphy and Ms Maureen Dwane, of the Boole Centre for Research in Informatics. Sincere thanks are due to Maureen for providing unstinting support in all matters relating to administration. Full financial support was generously provided by the Boole Centre for Research in Informatics at NUI, Cork.

The workshop website is http://www.bcrl.ucc.ie/CSP08/index.html, and more details of the workshop can be found there.
News from Latin America

by

Alfredo Viola

Instituto de Computación, Facultad de Ingeniería
Universidad de la República
Casilla de Correo 16120, Distrito 6, Montevideo, Uruguay
viola@fing.edu.uy

In this issue I present the report of the Third Latin-American Workshop on Cliques of Graphs by Flavia Bonomo and the call for papers of LAGOS’09 and CIBSI’09. At the end I present a list of the main events in Theoretical Computer Science to be held in Latin America in the following months.

Report of the Third Latin-American Workshop on Cliques of Graphs (by Flavia Bonomo)

The Third Latin-American Workshop on Cliques of Graphs was held from October 28-31, 2008, in Guanajuato, Mexico. Guanajuato is a magical little city located in the middle of Mexico, characterized by its underground streets, tunnels and narrow alleys. Among them, the Alley of the Kiss (el Callejón del Beso) has a tragic love legend between a poor miner, Carlos, and a rich girl, Ana, the Romeo and Juliet from Guanajuato. Guanajuato is also the seat of the International Cervantino Festival, a very important and famous festival plenty of culture, music, theatre and dance, which is held every year since 1972 and that draws artists and tourists from around the world. The workshop was hosted by the Centre for Mathematical Research (CIMAT), one of the most important research centres of the country. CIMAT has a very modern and comfortable infrastructure, and it is placed in Valenciana Mine, having a very beautiful view of the city. The
First Latin-American Workshop on Cliques of Graphs was held in Rio de Janeiro, Brazil, in 2002 and the second one took place in La Plata, Argentina, in 2006. For the third edition, about 30 researchers from Argentina, Brazil, Chile, Mexico, Switzerland and USA shared a really nice week of talks, problem discussions and fun. There were five invited talks: "The rectilinear crossing number of Kn: closing in (or are we?)", by Gelasio Salazar from Universidad Autónoma de San Luis Potosí, Mexico; "From Chordal to Helly circular-arc graphs", by Liliana Alcón, from Universidad Nacional de La Plata, Argentina; "The maximum number of edges in the line graph", by Bernardo Ábrego, from California State University, USA; "On colouring of graphs" by Célia P. de Mello, from Universidade Estadual de Campinas, Brazil, and "Why Cliques?", by Erich Prisner, from Franklin College, Switzerland. The program also included 18 contributed works and an interesting open problems session. Social activities included a welcome cocktail, involving Chilean wine, typical mezcal, and a good opportunity to meet old friends and make some new ones. The closing dinner was at one of the best restaurants in Guanajuato. Of course, during the dinner we enjoyed live music from the traditional mariachis. It was a very good job by the organizing committee, formed by Hernán González, Marisa Gutierrez, Paco Larrión, Martín Matamala, Miguel Pizaña, Jayme Szwarcfiter and Jorge Urrutia. The next edition of this workshop will be in 2010, in Rio de Janeiro, Brazil. I hope to see you there.

Call for Papers: LAGOS’09

The V Latin-American Algorithms, Graphs and Optimization Symposium (LAGOS’09) will be held in Gramado, Rio Grande do Sul, Brazil on November 3-7, 2009.

LAGOS is the union of two Latin American Conferences on these subjects: the GRACO (Brazilian Symposium on Graphs, Algorithms and Combinatorics) and the LACGA (Latin American Conference on Combinatorics, Graphs and Applications). The first GRACO was held in Fortaleza, Brazil, in 2001, and the second GRACO was in Angra dos Reis, Rio de Janeiro, Brazil, 2005. The LACGA was held in Santiago, Chile, in 2004. The first unified LAGOS (IV LAGOS) was held in Puerto Varas, Chile, 2007. The format of these meetings was similar. The proceedings of the accepted papers were published in Electronic Notes in Discrete Mathematics: Volume 7, for the first GRACO, Volume 19 for the second GRACO, Volume 18 for LAGCA, and Volume 30 for the IV LAGOS. Besides, special editions of Discrete Applied Mathematics were dedicated to these events: Volume 141 for the first GRACO, Volume 154 (13) for LACGA, and Volume 156 (7) for the second GRACO. The publication relative to the IV LAGOS is due to appear soon.

Papers presenting new and original research on the following (and related) top-
ics are sought: algorithms and data structures, analysis of algorithms, approximation algorithms, combinatorial optimization, complexity, computational biology, cryptography, enumerative combinatorics, graph theory, mathematical programming, operations research, order theory, polyhedral combinatorics and randomized algorithms.

The submission deadline is April 30, 2009, and the web page of the conference is http://www.inf.ufrgs.br/lagos09.

Call for papers: CIBSI’09
The fifth Ibero-American Congress on Information Security (CIBSI’09) will be held on November 16-18, 2009, in the city of Montevideo, Uruguay, and is organized by the University of the Republic (UDELAR) and the Universidad Politecnica de Madrid. The submission deadline is April 15, 2009, and the web page of the conference is http://www.fing.edu.uy/cibsi09.

Regional Events

NEWS FROM NEW ZEALAND

BY

CRISTIAN S. CALUDE

Department of Computer Science, University of Auckland
Auckland, New Zealand

1 Scientific and Community News

0. Conference.

1. The latest CDMTCS research reports are (http://www.cs.auckland.ac.nz/staff-cgi-bin/mjd/secondcgi.pl):

324. C.S. Calude. Incompleteness: A Personal Perspective, 06/2008


331. S. Figueira, J. Miller and A. Nies. Indifferent Sets, 08/2008


334. L. Staiger. On Oscillation-free $\varepsilon$-Random Sequences, 09/2008

335. J. Mielke. Refined Bounds on Kolmogorov Complexity for $\omega$-Languages, 09/2008


340. C. Müller and M. Kohlhase. Communities of Practice in Mathematical E-Learning, 11/2008


2 A Dialogue on Theoretical and Applied Computer Science with Professor Hermann Maurer

Professor Hermann Maurer is well known to the EATCS community of which he has been a prominent member for many years. He is Chairman of the Institute for
The Bulletin of the EATCS

Information Systems and Computer Media (IICM), Graz University of Technology, Vice-Chairman of the Board of Hyperwave AG and Chairman of the Advisory Board of Austria’s Competence Center for Knowledge Management (Know-Center). His webpage is www.iicm.edu/maurer.

Professor Maurer wrote more than 20 books and over 600 scientific articles, started or has been involved with a number of companies, supervised over 40 dissertations. He has obtained impressive results in both theory and applied computer science, so he is eminently qualified to compare theory from both inside and outside.

Professor Maurer has also written a number of Science Fiction books, a few available in English, see e.g. www.iicm.edu/Xperts.¹

Cristian Calude: Let us start this dialogue by asking you to reminiscence about your time as EATCS Bull. editor.

Hermann Maurer: I took this over from Maurice Nivat with No. 3 of the Bulletin, so we had to create the EATCS logo that still appears (after some 35 years!) on the first page of the Bulletin and also the picture of the automaton which always appears with the Council composition. When I was secretary, the work in EATCS intensified a lot, so the Bulletin grew from some 20 pages to its current volume. The fact that the volume increased rather than decreased is due to G. Rozenberg, who took over the editorship of the Bulletin after me. His imagination, the various columns and cartoons from Dardara have influenced the Bulletin up to today in a decisive manner. I am happy I was able to be an active member of the community for about ten years—after which my interests slowly changed to more applied areas—yet my allegiance to EATCS has never changed.

CC: You started your career by obtaining a doctorate in mathematics in 1965 under Professor Edmund Hlawka, with a dissertation entitled Rational Approximations of Irrational Numbers. Please comment two results from your Thesis.

HM: In the thesis I basically disproved the claim that using hyper-geometric series would give better approximations to irrational numbers than were obtainable by continued fractions or similar methods. Since it is a negative result, in the sense of lower bound, the proof was reasonably tedious.

CC: Then you worked for IBM in Vienna . . .

HM: Before I worked for IBM I was an assistant to Professor John Peck, then from the University of Calgary, Canada, one of the leading experts in programming languages. He was one of the inner group developing Algol 68. I

¹He recommends (for anybody interested) to start with the “Paranet”.

31
am indebted to him for showing me the beauty of exact (formal) definitions for programming languages. I then had almost a year a job as systems analyst with the Government of Saskatchewan, the first time I had a chance to apply computer solutions to large practical programs. On my return to Vienna I joined the IBM research Lab in Vienna, headed by the father of computer science in Austria, Professor Zemanek. I developed a first list processing compiler and then worked in the group that eventually developed VDL, the Vienna Definition Language. In this group I was apprentice to powerful brains like the one of Peter Lucas: under his (and Hans Bekic and Kurt Walk) the technique developed into what is today called Formal Methods Europe; it has survived four decades, something unheard of in computer science. Lest I am misunderstood: I was a tiny and unimportant part of this development, but I learnt a lot about formal methods.

CC: You have been an active—better still, hyper-active—member of the theoretical computer science community. In which areas have you worked?

HM: I continued my work in theoretical computer science when I re-joined the University of Calgary as assistant, later associate professor, in 1966. During my first years there I completed what was to be the first German pocket book on the theory (formal description) of programming languages: it turned into a best-seller. Nobody can still understand why this book sold some 50,000 copies at a time (1969) when there were certainly not yet 50,000 German speaking computer scientists. For a number of years I continued in formal languages and automata theory.

CC: How was theoretical computer science in the 70s?

HM: In the early 70s Formal Languages and Automata Theory and related areas (like undecidability results and such) where the central focus of theoretical research. Towards the middle of the 70s the theory of data-structures and algorithms gained more and more importance, and I started to work on this with a number of PhD students and post graduates. Some members of my group, like Professor Ottmann (now University of Freiburg), Professor Albert (now University of Würzburg), Professor Edelsbrunner (now Duke University) or Professor Welzl (now ETH) are just a few of my early students who eventually became more successful in this area than me.

CC: Please tell us your story about the famous MSW team?

HM: MSW stood for Maurer-Salomaa-Wood, with the person in the center, Salomaa, being also the central figure. However, we were really a great team: two of us would get together and try to find new proofs and results, the third one was the one who would do the proof-reading, trying to find holes in the proofs, etc.
We became particularly well-known by investigating L-systems and L-Forms, combining and extending ideas of the late Professors Ginsburg and Lindenmayer. Altogether we published some 50 journal contributions together and there was never the smallest amount of friction between us. Clearly, we have remained life-long friends.

**CC**: How easy/pleasant was to do joint work with Karel Culik?

**HM**: Karel Culik II has a brilliant mind. Working with him was not easy, however, since he was always ahead, but always also very intuitive. So when I worked with him I was never sure: am I not clever enough to understand what he says, or is he just arm-waving. And it was always both: he had this incredible talent of immediately seeing the rough solution of very complicated problems, but the first idea had to be refined often dramatically to finally give a watertight proof. The story of the DOS sequence equivalence problem, where he submitted \( n \) versions (\( n \) quite large), each not correct, is typical for him: he eventually, however, managed to overcome all gaps in the original proof-idea and solved this difficult problem.

**CC**: When and why did you switch from theory to applied computer science?

**HM**: My switch was fairly gradual, and much dictated by circumstances: first, formal languages, then algorithms, a bit of cryptography, then due to demand, applying this more and more to real-life problems, eventually reducing theory to just one of the things to do. It was also a pragmatic switch: it was much easier to get funding for applied projects than for very theoretical ones.

**CC**: Are there different skills required in doing theory vs. applied computer science?

**HM**: I think to do good work in applied areas one does need a solid grounding in theory. The connection is not always clear to see, but the correlation is. There is, however, one basic difference: to be successful in applied work you have to be able to lead substantial teams, something less important for theoretical work.

**CC**: Can you compare your most important results in theory and applied computer science? Which gave you more satisfaction? Why?

**HM**: This is a very hard question. A good result in theory gives you the satisfaction of having solved a difficult puzzle that nobody has solved before, the satisfaction that you see things that were overlooked before. Applied projects give you the satisfaction that what you have done may be used by a large number of people, may lead to the formation of companies, to new jobs, and also to some recognition as entrepreneur.
CC: Last week you gave a very interesting lecture in Auckland entitled *Theory is important but dangerous*. What was its main message?

HM: The answer is almost in the title. We do need theory, both to train students and to solve the theoretical basis for important problems. Examples are numerous: from cryptography to picture compression, from the organisation of data to computer vision, etc. At the same time, theory can be dangerous, mainly because negative results tend to scare people away from trying to solve problems whose exact solution may be very hard, but where there are very good approximate solutions.

CC: As in mathematics, in theoretical computer science beauty is a guiding principle as well as an ultimate indicator of value. Does it make sense to ask “How really useful is theoretical computer science”?

HM: No, the question does not make sense. If we did not have, did not teach, did not research in theory, we would not get anywhere in applications.

CC: What would be your principles in designing a balanced curriculum in computer science?

HM: A balanced curriculum must include a solid basis in mathematics and theoretical computer science, but must also teach more applied areas in a generic way. The emphasis is here on “generic”. One must not study or work with concrete systems (software-packages) but one must make clear the underlying principles. Only in this way it is possible to make sure that the knowledge acquired does not obsolete too fast.

CC: Why computer science seems to have lost its appeal to the young generation?

HM: I really don’t think it has. That numbers have dropped in some places is due to the negative PR around the “bust of the internet bubble” that suddenly created the wrong impression that computer science would not be important any more. However, this is definitely not so. The world needs many more IT specialists than are currently produced.

CC: You wrote extensively on the future of technology using metaphors such as telepathy and teleportation. Tell us please more about these unusual writings?

HM: Basically, any technology that is far beyond our current one looks like a miracle. This statement is often attributed to Arthur Clarke, the SF writer, but goes actually much further back to around 1850! Just consider: if you had used a

---

2November 7, 2008.
digital camera a few hundred years ago you certainly would have been burnt at the stake. Things like telepathy or telekinesis that sound like impossible or miracles today will be implemented with suitable technologies before too long. Telepathy is almost a reality due to new brain-computer interfaces, and things like telekinesis will be handled (I quote Kurzweil) by swarms of nanobots. So: it is fun to write things that sound like SF but might well turn into reality at some stage, see XPERTS in www.booklocker.com or my home-page www.iicm.edu/maurer.

CC: How dangerous is Google?

HM: Google is THE largest data-mining company we have in the world. It knows more about people and organisations than any intelligence agency, and there is no control via data-protection laws at this point in time. This gives more power into the hand of one particular company than is good for all of us. Actions will have to be taken to curtail those powers, and I know that such actions are on the way, without ruining the great advantages Google has for all of us due to its marvellous services, nor without ruining the value of the company for Google share holders.

CC: What are your next decade projects?

HM: Open access scientific journals, helping to develop quality-improved community encyclopaedias akin Wikipedia, and assuring that when we look for something on the Internet we don’t find millions of pieces of information, but the knowledge we have been looking for.

CC: Thank you very much.
The EATCS Columns
The Algorithmic Game Theory Column

by

Mario Mavronicolas

Department of Computer Science, University of Cyprus
75 Kallipoleos St., CY-1678 Nicosia, Cyprus
mavronic@cs.ucy.ac.cy

The Algorithmic Game Theory Column of BEATCS is now embarking on exposing the Theoretical Computer Science community to the most exciting and inspirational results in the young field of Algorithmic Game Theory. Roughly speaking, Algorithmic Game Theory focuses on the intersection of Game Theory and Algorithms, two intellectual fields that have shaped much of the second half of the twentieth century. Algorithmic Game Theory has numerous applications in many other fields including Networking and Artificial Intelligence, and a potential rapidly increasing with the grow of the Internet.

In a few words, the mission of Algorithmic Game Theory is to examine problems from Game Theory and Mathematical Economics through the lens of the Theory of Algorithms. Although Game Theory has succeeded in answering existential questions about its own concepts (with the famous Nash equilibrium as the prime example), much less was known about ten years ago about the corresponding computational questions. One of the currently major trends is to reveal the algorithmic foundations of game-theoretic concepts (such as Nash equilibrium, market equilibrium or the core). A second major trend lies around the most influential Price of Anarchy introduced in the seminal work of Koutsoupias and Papadimitriou from 1999, a popular measure of the inefficiency of a game-theoretic solution concept. Another major trend pioneered by Nisan and Ronen in their seminal work from 1999 is coined as Algorithmic Mechanism Design: how can one design a game with a desirable outcome?

This newborn column aims at popularizing the most exciting developments in this fascinating field. Indeed, the first article contributed by George Christodoulou and Elias Koutsoupias offers the basics on the Price of Anarchy under an extremely motivating perspective.

Concluding, I would like to warmly welcome you to the column. I encourage you to submit articles and other material. I especially welcome surveys, collec-
tions of interesting open problems or any other material contributing inspiration and novel insights into the field. I am also interested in reviews of the main related conferences or other significant events. Please contact me with your most interesting proposals!

**MECHANISM DESIGN FOR SCHEDULING**

George Christodoulou  
*Max-Planck-Institut für Informatik, Saarbrücken, Germany.*  
gchristo@mpi-inf.mpg.de

Elias Koutsoupias*  
*Department of Informatics, University of Athens, Greece*  
elias@di.uoa.gr

**Abstract**

We consider mechanism design issues for scheduling problems and we survey some recent developments on this important problem in Algorithmic Game Theory. We treat both the related and the unrelated version of the problem.

1 The scheduling problem

The problem of scheduling unrelated machines [21, 14] is one of the most fundamental algorithmic problems: There are $n$ machines and $m$ tasks and machine $i$ can execute task $j$ in time $t_{ij}$. These times can be totally unrelated (thus the name of the problem). The objective is to allocate the tasks to machines to minimize the makespan (the time needed to finish all tasks). Thus the output is simply a partition of the $m$ tasks into $n$ sets. A convenient way to express it is to use indicator variables $x_{ij} \in \{0, 1\}$: $x_{ij}$ is 1 iff task $j$ is allocated to machine $i$. Each task $j$ is allocated to exactly one machine, therefore we must have $\sum_{i=1}^{n} x_{ij} = 1$ for every $j$. With this notation, the computational problem can be expressed more precisely: given $n \times m$ values $t_{ij}$, find appropriate $x_{ij} \in \{0, 1\}$ which satisfy these constraints and minimize $\max_{j=1}^{m} \sum_{i=1}^{n} x_{ij} t_{ij}$.

>From the traditional algorithmic point of view, the unrelated machines scheduling problem is one of the most important open problems. We know that the

*Partially supported by IST-15964 (AEOLUS) and IST-2008-215270 (FRONTS).

*We opt for the game-theoretic notation here and we denote the number of machines and tasks with $n$ and $m$ respectively. In the scheduling literature, they usually use the opposite notation.
problem is NP-hard; it is even NP-hard to approximate it within \(3/2\) [21]; this lower bound applies also to some special cases [11]. On the positive side, there is a polynomial-time approximation algorithm with approximation ratio 2 [21]. Closing the gap between the lower and upper bounds on the approximation ratio remains a long-standing major algorithmic problem.

There are many interesting variants of the problem. When, for example, the times \(t_{ij}\) are inversely proportional to the speed of the machine, that is, when there are speeds \(s_i\) and times \(p_j\) such that \(t_{ij} = p_j / s_i\), we have the special case of the problem called the related machines scheduling problem. Also, when we allow a task to be split across the machines, which is to say that \(x_{ij}\) are nonnegative reals instead of integers, we call this the fractional scheduling problem. The computational complexity of these problems is completely settled: There is a polynomial time approximation scheme (PTAS) [13] for the related machines problem and a fully-PTAS (FPTAS) [15] when the number of machines is fixed; the general case is strongly NP-complete, so we don’t expect to find an FPTAS unless \(P=NP\). For the fractional version of the problem, there is a polynomial-time algorithm (because it can be expressed as a linear program).

Nisan and Ronen in their seminal work [27, 28] which started the area of Algorithmic Mechanism Design considered the unrelated machines problem from a game-theoretic point of view: suppose that each machine \(i\) is a rational agent who is the only one knowing the values of row \(t_i\). Suppose further that the machines want to minimize their execution time. Without any incentive, the machines will lie in order to avoid getting any task. To coerce the machines to cooperate, we pay them to execute the tasks. The payments do not have to be proportional to the execution times, but can be arbitrary functions. The combination of the algorithmic problem of allocating the tasks to machines together with the incentives in the form of payments is called a mechanism. In this article, we survey recent developments in this area of mechanisms for the scheduling problem.

We consider direct revelation mechanisms with dominant truthful strategies. Direct revelation means that the players—who know the mechanism in advance—declare their hidden values to the mechanism which collects the values and computes an allocation of tasks and appropriate payments to the players. In such a mechanism, a player may have an incentive to lie and declare values other than his true values. If the mechanism is such that, independently of the values of the other players, a player has no incentive to lie, we say that the mechanism is truthful (with dominant truthful strategies). These mechanisms are very desirable and easy to be implemented since there is no reason for machines to strategize. There other weaker notions of truthfulness but we don’t consider them in this note.
2 Mechanism design

The mechanism design setting for scheduling is a special case of the social choice problem. We define here the more general framework so that we can place the scheduling problem within the general picture. In the social choice setting, there are \( n \) players and a set of possible outcomes \( A \) which in most cases is considered finite. The players may value the outcomes differently. For each player there is a valuation function \( v_i : A \to \mathbb{R} \) which gives the value of player \( i \) for every outcome. The goal of the mechanism designer is to implement a social choice function \( f \) which assigns a desirable outcome to every set of valuation functions. For example, a social choice function selects the outcome for which the median of the valuations of the players is maximum. More generally, for every set of valuation functions there may be a set of desirable outcomes and the mechanism designer wants to implement a social choice function which selects one of these outcomes.

When we consider problems with finitely many possible outcomes, we can recast the above in a more familiar notation. A mechanism design problem with \( n \) players and \( k = |A| \) outcomes is defined by a subset \( D \) of \( \mathbb{R}^{n \times k} \). We call \( D \) the domain of the mechanism design problem. A social choice function \( f \) is simply a function from \( D \) to \( \{1, \ldots, k\} \) (or more generally to the collection of subsets of \( \{1, \ldots, k\} \)).

An instance of a mechanism design problem is simply a point of \( D \). A concrete way to represent a point of \( D \) is by a real-valued \( n \times k \) matrix \( v \). Each player \( i \) knows the values of row \( \hat{v}_i \); this is private information and it is not known to the mechanism. In a direct-revelation mechanism, each player \( i \) declares values \( v_i \) of row \( i \). These values may be different than \( \hat{v}_i \). The declarations of the players form a matrix \( v \). The mechanism takes as input this matrix \( v \) and computes two quantities: an outcome \( o = o(v) \in \{1, \ldots, k\} \) (the outcome of the mechanism) and payments \( p = p(v) \in \mathbb{R}^n \) for the players. The payoff of player \( i \) is the value of the original row at the outcome minus the payment: \( \hat{v}_{i,o} - p_i \).

In summary, a mechanism design problem is defined by a set \( D \) of \( n \times k \) real matrices (a subset of \( \mathbb{R}^{n \times k} \)) and a function \( f \) from \( D \) to subsets of \( \{1, \ldots, k\} \). The mechanism designer must come up with an outcome function

\[
o : D \to \{1, \ldots, k\},
\]

and a payment scheme

\[
p : D \to \mathbb{R}^n,
\]

such that the outcome \( o \) is a desirable outcome \((o(v) \in f(\hat{v}))\) and the payment scheme induces the players to declare values which produce desirable outcomes.

The mechanism is truthful when the outcome and payment functions are such that the players gain nothing by not declaring their true values, i.e., the objective
of each player $i$, for every declared values $v_{-i}$ of the other players, is maximized when player $i$ declares $v_i = \hat{v}_i$. This notion of truthfulness is called dominant-strategy truthfulness since declaring the true values is a dominant strategy for each player.

The Revelation Principle states that for every mechanism there is an equivalent truthful mechanism which on the same input $\hat{v}$ has the same outcome and payments. This frees us to consider only truthful mechanisms.

Here are a few typical mechanism design domains:

**Example** (The unrestricted domain): One of the most natural mechanism design problems for $n$ players and $k$ outcomes is when $D$ is the whole $\mathbb{R}^{n \times k}$ space.

**Example** (The combinatorial auction domain): The outcomes are all allocations of $m$ items to $n$ players; there are $k = n^m$ possible allocations. The domain is defined by values $v_{i,x}$, one for each player $i$ and for each allocation $x$. An allocation $x$ is defined by a matrix where $x_{ij}$ is a variable indicating whether task $j$ is allocated to player $i$. The values $v_{i,x}$ satisfy the natural restriction that the valuation of a player $i$ depends only on the items allocated to him: $v_{i,x} = v_{i,x'}$ when the two allocations agree on player $i$ (that is, when $x_{ij} = x'_{ij}$). In economic terms this condition says that there are no externalities. Another natural restriction is that the value of a player can only increase when he gets additional items ($v_{i,x} \geq v_{i,x'}$ when $x_{ij} \geq x'_{ij}$ for $j = 1, \ldots, m$). The values are also nonnegative and they are exactly 0 when a player is allocated no item.

For example, for $n = 3$ players and $m = 2$ items the domain $D$ contains the points of $\mathbb{R}^{3 \times 2}$ which are of the form:

<table>
<thead>
<tr>
<th>$u_{1,12}$</th>
<th>$u_{1,1}$</th>
<th>$u_{1,2}$</th>
<th>$u_{1,2}$</th>
<th>$u_{1,2}$</th>
<th>$u_{1,2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_{2,1}$</td>
<td>$u_{2,1}$</td>
<td>$u_{2,1}$</td>
<td>$u_{2,1}$</td>
<td>$u_{2,1}$</td>
<td>$u_{2,1}$</td>
</tr>
<tr>
<td>$u_{3,1}$</td>
<td>$u_{3,1}$</td>
<td>$u_{3,1}$</td>
<td>$u_{3,1}$</td>
<td>$u_{3,1}$</td>
<td>$u_{3,1}$</td>
</tr>
</tbody>
</table>

where the values are nonnegative and $u_{1,12} \geq u_{1,1}$ and $u_{1,12} \geq u_{1,2}$.

For the special case of the single-item auction, the matrix is a diagonal one where the value $u_{i,i}$ is the valuation of player $i$ for the item.
Example (The unrelated machines domain): This is a special case of the combinatorial auction when the domain is additive. It is also a cost game (as opposed to a payoff one). The outcomes again are all allocations of $m$ tasks to $n$ players ($k = n^m$). The domain is defined by $v_{i,s} = \sum_{j=1}^{m} x_{ij}$, $t_{ij}$. For example, for $n = 2$ players and $m = 2$ tasks the domain $D$ contains the points of $R^{2 \times 4}$ for which are of the form:

\[
\begin{pmatrix}
t_{11} + t_{12} & t_{11} & t_{12} & 0 \\
0 & t_{22} & t_{21} & t_{21} + t_{22}
\end{pmatrix}
\]

where $t_{ij}$'s are nonnegative.

2.1 Known mechanisms

Given a domain, one can ask whether there are any (truthful) mechanisms. If we view the payments as the means to implement a social choice function, we can rephrase the question: For which social choice functions are there payment functions so that the resulting mechanism is truthful? In this way we focus on the social choice function. For example, for the single-item auction domain, are there payment functions for a mechanism to allocate the item to the player with maximum (private) value? with the second maximum value? As we will discuss soon, the answer to the first question is positive and to the second question negative.

There are few mechanisms that are known to be truthful and the best-studied one is the VCG mechanism [31, 8, 12].

Example (The VCG mechanism and affine maximizers): The VCG mechanism implements the social choice function of selecting the outcome (column) with the maximum total value:

\[
f(v) = \arg\max_{j \in \{1, \ldots, k\}} \sum_{i=1}^{n} v_{ij}.
\]

A generalization of this mechanism is the affine maximizer which weights with positive multipliers $\lambda$, the values of each player (row) $i$ and add a constant $\gamma_j$ to the value of each outcome (column) $j$:

\[
f(v) = \arg\max_{j \in \{1, \ldots, k\}} \sum_{i=1}^{n} \lambda_i v_{ij} + \gamma_j.
\]
The VCG mechanism is truthful for every domain. The payments (for the general domain) align the objective of each player \( l \) with the social choice function. This can be achieved when the payments are

\[
p_l(v) = -\frac{1}{\lambda_l} \left( \sum_{j=1}^{\lambda_l} \lambda_i v_{ij} + \gamma_j \right).
\]

Player \( l \) wants the mechanism to select an outcome \( j \) which maximizes \( v_{lj} - p_l(v) = \frac{1}{\lambda_l} \sum_{i=1}^{\lambda_l} \lambda_i v_{ij} + \gamma_j \). This is the same expression with the argmax expression above and shows that the player’s objective is achieved at the social choice function. The VCG has slightly different payments: Because these payments may be negative, the VCG mechanism adds appropriate values to the payment of each player that depend only on the values of the other players (this keeps the player truthful).

Another interesting class of mechanisms for the scheduling problem are the task independent mechanisms: Each task is allocated independently of the remaining tasks. Not all task-independent mechanisms are truthful. Task-independent mechanisms are special cases of threshold mechanisms:

Example (Threshold mechanisms): A threshold mechanism for the scheduling domain is one for which there are threshold functions \( h_{ij} \) such that the mechanism allocates item \( j \) to player \( i \) if and only if \( v_{ij} \geq h_{ij}(v_{-i}) \). What distinguishes these mechanisms from general mechanisms is that the thresholds depend only on the values of the other players but not on the other values of the player himself. It is not true in general that every set of functions \( h_{ij} \) defines a legal mechanism, as they have to be consistent between them. In particular, the threshold functions should be such that every item \( j \) is allocated to exactly one player. In other words, exactly one of the constraints \( v_{ij} \geq h_{ij}(v_{-i}) \), for \( i = 1, \ldots, n \), should be satisfied.

\section{Truthfulness}

One of the central questions in mechanism design is to find a nice characterization of truthful mechanisms. In algorithmic terms, we want to determine which algorithms are implementable, i.e., for which algorithms for the scheduling problem there exist payments that make the players truthful. It should be clear that for rich domains, such as the scheduling domain, not all algorithms are truthful. In fact, it seems that the set of truthful algorithms is very limited, but whether this is the case
or not is perhaps the most outstanding open problem in algorithmic mechanism
design:

Open Problem. Characterize the set of truthful mechanisms for scheduling.

But what kind of characterization we seek? We are going to see that we do
have a necessary and sufficient condition, the so-called Monotonicity Property.
But we want a characterization which is more than a necessary and sufficient con-
dition. An important result in the area of mechanism design, Roberts’ Theorem
[16], shows exactly the type of characterization we seek. Roberts’ Theorem ap-
plies to the unrestricted domain and states that the only truthful mechanisms for
this domain of \( k \geq 3 \) outcomes are the affine maximizers. In a sense, this is a very
disappointing result, because it says that only very simple algorithms can be im-
plemented. The question becomes much more interesting for restricted domains
and in particular for the auction and scheduling domain. It is a simple observation
that when we restrict the domain the set of available mechanisms can only become
richer. More precisely, for domains \( D \subset D' \), every truthful mechanism for \( D' \) is
also a truthful mechanism for \( D \).

We discuss below the Monotonicity Property which a simple necessary and
sufficient condition for truthfulness. This is true for every convex domain, but we
restrict the discussion to the scheduling domain.

Definition 3.1 (Monotonicity Property). An allocation algorithm is called mono-
tone if it satisfies the following property: for every two sets of tasks \( t \) and \( t' \) which
differ only on machine \( i \) (i.e., on the \( i \)-th row) the associated allocations \( x \) and \( x' \)
satisfy

\[
(x_i - x'_i) \cdot (t_i - t'_i) \leq 0,
\]

where \( \cdot \) denotes the dot product of the vectors, that is, \( \sum_{j=1}^{m} (x_{ij} - x'_{ij})(t_{ij} - t'_{ij}) \leq 0 \).

The property, which sometimes in the literature is called weak monotonicity,
essentially states that when we increase the times of the tasks for machine \( i \), the
allocation for the machine can only become smaller. Notice that the Monotonicity
Property involves only the allocation of one player (the \( i \)-th player).

Theorem 3.2 (Saks and Yu). A mechanism is truthful if and only if its allocations
satisfy the Monotonicity Property.

To establish that the Monotonicity Property is necessary for truthfulness is
easy (it was done for example in [28]) and we show it below. Saks and Yu showed
that it is also a sufficient condition. In fact, they showed a much more general
result: the property is sufficient for every convex domain; this includes the unre-
stricted domain and the combinatorial auction domains.
The Bulletin of the EATCS

To show that the property is necessary condition for truthful mechanisms, we observe that the payments cannot depend directly on the declaration \( t_i \) of player \( i \), but indirectly through the selected outcome \( x(t) \) and the declarations \( t_{-i} \) of the other players, that is, \( p_i(t) = p_i(x(t), t_{-i}) \). To see this, suppose that there exist \( t, t' \) such that \( x(t, t_{-i}) = x(t', t_{-i}) \), but \( p_i(t, t_{-i}) < p_i(t', t_{-i}) \). Then the player whose true processing times are \( t \) has incentive to declare falsely that its processing times are \( t' \) in order to increase his utility, as we have \( p_i(t, t_{-i}) - \sum_{j=1}^{n} t_{i,j} < p_i(t', t_{-i}) - \sum_{j=1}^{n} t_{i,j} \), contradicting the truthfulness of the mechanism.

When player \( i \) has valuations \( t_i \), he has no incentive to declare \( t'_i \) when

\[
t_i x_i - p_i(x_i, t_{-i}) \leq t'_i x'_i - p_i(x'_i, t_{-i})
\]

Similarly, when we inverse the roles of \( t \) and \( t' \), we have

\[
t'_i x'_i - p_i(x'_i, t_{-i}) \leq t_i x_i - p_i(x_i, t_{-i})
\]

Now if we add the above inequalities and take into account that the instances differ only on the \( i \)-th player, that is, \( t'_i = t_{-i} \), we get the Monotonicity Property.

The implications are that we don’t have to consider at all the payment algorithm. This transforms the problem from the realm of Game Theory to the realm of Algorithms. To prove lower bounds or to design good mechanisms, we can completely forget about mechanisms, payments, truthfulness etc, and simply focus on the subclass of monotone allocation algorithms.

The Monotonicity Property has a straightforward geometric form. For simplicity, let us consider 2 tasks and consider the space of possible valuations for a particular machine \( i \). The generalization to more tasks is straightforward. Fix the values \( t_{-i} \) of the remaining players. For every \((t_1, t_2)\), let us consider how a mechanism which satisfies the Monotonicity Property allocates the tasks. In particular, let \( R_{v_1v_2} \) denote the set of inputs of player \( i \) for which the mechanism has allocation \((x_1, x_2)\) for the \( i \)-th player. The Monotonicity Property is equivalent to the constraint that the boundary between \( R_{v_1v_2} \) and \( R_{v'_1v'_2} \) is of the form \((x'_1 - x_1)t_1 + (x'_2 - x_2)t_2 = 0\). Since the allocation variables \( x_1 \) and \( x_2 \) are 0-1, the boundaries have very specific slopes. Therefore the allocation of the mechanism should have one the 2 forms of Figure 1.

In other words, a mechanism is truthful if and only if it partitions the \( R^{v_1v_2} \) space so that the appropriate lower dimensional cuts have the form of Figure 1. Thus characterizing the truthful mechanisms amounts to characterize the partitions of \( R^{v_1v_2} \) that have specific lower dimensional cuts.

Affine minimizers are the special class of algorithms for which the boundaries in Figure 1 are linear functions of the values of the other players. The diagonal part in the picture exist if and only if the additive constants \( \gamma_j \) are not all equal.
On the other hand, threshold mechanisms are exactly those whose diagonal part has 0 length (i.e., the partition is defined by orthogonal hyperplanes).

A recent paper by Dobzinski and Sundararajan [10] gives a simple characterization of mechanisms for 2 machines. They consider only mechanisms which have bounded approximation ratio with respect to makespan and they show that only task-independent mechanisms can be truthful. In [7], a more complete characterization was given which is independent of the approximation ratio: for 2 machines only affine minimizers and threshold algorithms can be truthful.

In the next 2 sections we consider positive and impossibility results for the unrelated machines problem. In the last section, we discuss positive results for the related machines version.

4 Upper bounds for the unrelated case

There are only a few positive results which give approximation algorithms for the unrelated machines scheduling problem. We discuss most of them here:

Deterministic mechanisms: Nisan and Ronen [28] gave a mechanism that is $n$-approximate. The mechanism is essentially the VCG, i.e., it assigns job $j$ to the machine with minimum $t_{ij}$. It runs independent second-price auctions per item, which is equivalent to the VCG because the valuations in the scheduling domain are additive.

Randomized mechanisms: There are two major notions of truthfulness for randomized mechanisms: universally truthful and truthful in expectation mechanisms. A universally truthful mechanism is a probability distribution over truthful deterministic mechanisms; this means that even when the players know the

---

†This holds only for decisive mechanisms, that is mechanisms where all allocations are possible; non-decisive algorithms are not very natural and among their properties is that they have unbounded approximation ratio.
outcome of the random choices (coins), they have no incentive to lie. This is in contrast to the truthful in expectation mechanisms where players have no incentive to lie before the random choices but they may have incentive to lie after the random choices.

Nisan and Ronen [28] suggested the following 1.75-approximate randomized mechanism for 2 machines. The mechanism is a universally truthful one and it works as follows: For every task \( j \), with probability \( 1/2 \) the algorithm gives the item to the minimizer of \( \min\{t_{1j}, \frac{4}{3} t_{2j}\} \), and with \( 1/2 \) to the minimizer of \( \min\{t_{2j}, \frac{4}{3} t_{1j}\} \). Mu’alem and Schapira [24] extended the mechanism for \( n \) machines which gives approximation ratio \( 1.875n \).

Recently, the result of Nisan and Ronen for 2 machines was improved by Lu and Yu [22] who gave a 1.67-approximation mechanism; they improved this later [23] to 1.59.

**Fractional mechanisms:** Christodoulou et al. [5] gave an algorithm for the fractional version of the problem which allocates each task independently. The fractions of task \( j \) assigned to machines 1, 2, \ldots, \( n \) are inversely proportional to the squares of the execution times of task \( j \). For example, for 2 machines the allocation of task \( j \) is given by

\[
x_{1j} = \frac{t_{2j}^2}{t_{1j}^2 + t_{2j}^2}, \quad x_{2j} = \frac{t_{1j}^2}{t_{1j}^2 + t_{2j}^2}.
\]

The mechanism has approximation ratio \( \frac{n+1}{2} \) and this is optimal for task-independent mechanisms.

**Restricted Domain mechanisms:** Lavi and Swamy [20] studied two cases where the valuation domain is restricted. Instead of allowing \( t_{ij} \) to get any positive real values, they restrict the values to 2: low and high. They show that in such domains there exist algorithms with constant approximation ratio, in contrast to general domains where the current best upper bounds are linear with respect to \( n \).

These domains are not convex and Theorem 3.2 of Saks and Yu [30] does not apply. Instead a more complicated property, the cycle monotonicity property [29], is necessary and sufficient for this domain (and every other domain): The cycle monotonicity property considers closed paths of inputs and requires that the sum of a certain expression is nonnegative over every cycle. The Monotonicity Property is the special case when the cycles have length 2.

When the tasks are allowed to have different low and high values, Lavi and Swamy gave a 3-approximate algorithm which is truthful in expectation. The algorithm computes the optimal fractional solution, it transforms it to a cycle-monotone fractional allocation, and finally rounds it using randomized rounding.
When all tasks have the same low and high values, an even better result is possible: they gave a 2-approximate deterministic cycle-monotone algorithm based on maxflow.

5 Lower bounds for the unrelated case

In this section we summarize the main impossibility results and techniques for the unrelated machines problem. It so happens that all the lower bounds for this problem (deterministic, fractional, and randomized) are based on the restrictions of truthfulness and they hold independently of computational complexity considerations. In other words, the lower bounds apply even to exponential-time algorithms.

Nisan and Ronen [28] gave a lower bound of 2 for any truthful deterministic mechanism for 2 machines. Christodoulou et al. [6] improved the lower bound to $1 + \sqrt{2} \approx 2.41$ for 3 or more machines, and Koutsoupias and Vidali [17] to $1 + \phi \approx 2.61$ for $n$ machines where $n$ is arbitrarily large. It is a major open problem to close the gap between the lower and the upper bound.

Conjecture (Nisan and Ronen). No mechanism has approximation ratio better than $n$.

Mu’alem and Schapira [24] gave a lower bound of $2 - \frac{1}{n}$ for randomized truthful in expectation mechanisms (which also applies to universally truthful mechanisms). Christodoulou et al. [5] showed that the same bound holds even for fractional domains. Notice that while for deterministic and fractional mechanisms we have tight bounds for 2 machines, for randomized mechanisms there is still a gap between the lower bound of 1.5 and the upper of 1.59. It is an interesting open problem here is to close this gap. Interestingly, even for the restricted domain of two values, Lavi and Swamy [20] showed a lower bound of $11/10$.

In the next subsections we discuss the basic ideas behind the lower bounds for deterministic mechanisms. We don’t consider randomized and fractional settings, but the main ideas are similar (although sometimes more complicated) [24, 5].

5.1 The case of 2 machines

Recall that every truthful mechanism is monotone. A useful tool that comes out of the Monotonicity Property and is used implicitly or explicitly in most of the lower bound proofs is the following.

\footnote{It is almost trivial to see that any lower bound for $n$ machines applies to the case of more than $n$ machines.}
Lemma 5.1. Let $t$ be the input matrix and let $x = x(t)$ be the allocation produced by a truthful mechanism. Suppose that we change only the processing times of machine $i$ in such a way that $t'_{ij} > t_{ij}$ when $x_{ij} = 0$, and $t'_{ij} < t_{ij}$ when $x_{ij} = 1$. A truthful mechanism does not change the allocation to machine $i$, i.e., $x_i(t') = x_i(t)$.

Proof. By the Monotonicity Property 3.1, we have that

$$\sum_{j=1}^{m} \left( t_{ij} - t'_{ij} \right) \left( x_{ij}(t) - x_{ij}(t') \right) \leq 0.$$ 

Observe that all terms of the sum are nonnegative (by the premises of the lemma). The only way to satisfy the inequality is to have all terms equal to 0, that is, $x_{ij}(t) = x_{ij}(t')$.

Now we will use this lemma to get easily a lower bound of 2, which first appeared in [28].

Theorem 5.2. Any truthful mechanism has approximation ratio of at least 2 for two or more machines.

Proof. Suppose that we have an instance with $n = 2$ and $m = 3$ and $t_{ij} = 1$, for all $i, j$. Any allocation algorithm can either allocate all tasks to a single machine (say the first one), or partition them (say the first two tasks to the first machine and the third task to the second machine). In the former case, we apply Lemma 5.1 to the first player (where the star symbol indicates allocation, and $\epsilon$ is an arbitrarily small positive number):

$$t = \begin{pmatrix} 1^* & 1^* & 1^* \\ 1 & 1 & 1 \end{pmatrix} \Rightarrow t' = \begin{pmatrix} 1 - \epsilon^* & 1 - \epsilon^* & 0^* \\ 1 & 1 & 1 \end{pmatrix}.$$ 

The resulting assignment on $t'$ has approximation ratio of $2\frac{1 - \epsilon}{1 + \epsilon} \approx 2$. In the latter case, we apply Lemma 5.1 to the second player:

$$t = \begin{pmatrix} 1^* & 1^* & 1^* \\ 1 & 1 & 1^* \end{pmatrix} \Rightarrow t' = \begin{pmatrix} 1^* & 1^* & 1 \\ 1 + \epsilon & 1 + \epsilon & 0^* \end{pmatrix}.$$ 

The resulting assignment on $t'$ has approximation ratio of $2\frac{1}{1 + \epsilon} \approx 2$.

5.2 The case of 3 or more machines

There is a qualitative difference between the case of 2 machines and the case of 3 or more machines. For 2 machines, the allocation of a player completely determines the allocation of the other player. This is not true for more than 2 players and it complicates the situation.
There are basically two approaches one can follow to prove a lower bound. One approach is to provide a global characterization of all possible mechanisms, such as Roberts’ Theorem. This approach however requires the solution of the characterization problem which is potentially a more difficult problem. For example, Christodoulou et al. [7] use this approach to extend the lower bound of 2 [27] to instances with only 2 tasks.

The other approach is to use an appropriately chosen subset of the input instances. The Monotonicity Property implies some relations between the allocations of these instances. We can use them to show that one of the instances has high approximation ratio. A typical application of this approach is the lower bound of 2.41 [6] and 2.61 [17]. In [6] as we will see later the set of instances is small. It consists of instances of 2 and 3 machines respectively and no more than 5 tasks. In [17], they use the same principles but apply them in an infinite subset of inputs using a double induction to keep track of how all these allocations depend on each other.

We sketch here the proof of the 2.41 lower bound.

Theorem 5.3. Any truthful mechanism has approximation ratio of at least $1 + \sqrt{2}$ for three or more machines.

The general idea of the proof is the following: We start with the set of tasks

$$
\begin{bmatrix}
0 & \infty & a & a \\
\infty & 0 & a & a \\
\infty & \infty & 0 & a & a
\end{bmatrix},
$$

for some parameter $a > 1$. This set of tasks essentially admits two distinct allocations (up to symmetry). This is true because the first three tasks need to be assigned to a single machine by any mechanism with bounded approximation ratio. For each allocation, we increase or decrease some values appropriately. Then is is shown that in order to keep the approximation ratio low (below $1 + a$), the following set of tasks must have the allocation indicated by the stars (in which the first machine gets both tasks 4 and 5):

$$
\begin{bmatrix}
0^* & \infty & \infty & 1^* & 1^* \\
\infty & 0^* & \infty & a & a \\
\infty & \infty & 0^* & a & a
\end{bmatrix}.
$$

Finally, the input of the first player is modified as the following matrix indicates. By using a lemma that is similar in spirit to Lemma 5.1, but in addition takes into account the fact that there is a unique way to allocate the first three tasks, we get the allocation

$$
\begin{bmatrix}
a^* & \infty & \infty & 1 - \epsilon^* & 1 - \epsilon^* \\
\infty & 0^* & \infty & a & a \\
\infty & \infty & 0^* & a & a
\end{bmatrix}.$$
This allocation has an approximation ratio of $\frac{2+\epsilon}{\alpha}$, for arbitrarily small value of $\epsilon$. Taking into account that the ratio is at most $1+\alpha$, we get the theorem.

## 6 Related machines

In this section, we consider the important special case of the scheduling problem, the related machines version. In this setting, the processing times of the tasks are $p_1 \geq \ldots \geq p_m$, while the machines have speeds $s_1, \ldots, s_n$. Given an assignment of the jobs to the machines, let $w_i$ denote the workload assigned to machine $i$. The makespan $C(w, s)$ is $\max_i \frac{w_i}{s_i}$. Monotonicity for this special case is very simple.

An algorithm is monotone (truthful) when it has the following property: when we decrease the speed of a machine $i$, keeping all other speeds the same, the new workload on machine $i$ can only decrease.

The mechanism design version of the problem was first studied by Archer and Tardos [3]. It is a very important problem in Algorithmic Mechanism Design, because it’s a typical single-parameter problem, which means that each player has only one real private value and his objective is proportional to this value (for a precise definition see Chapters 9 and 12 of [26]). Such problems were studied extensively by Myerson [25]. Furthermore, the optimal allocation is monotone and therefore truthful, but it cannot be computed in polynomial time unless $P=NP$. It is therefore an appropriate example to explore the interplay between truthfulness and computational complexity. It is a major open problem whether a deterministic monotone PTAS exists for this problem\footnote{We assume that a mechanism runs in polynomial time when both the allocation algorithm and the payment algorithm run in polynomial time.}. A very recent breakthrough result [9] shows that there exists a randomized truthful-in-expectation PTAS.

In contrast to the scheduling problem of unrelated machines, in this special case there exist truthful mechanisms that output an optimal allocation. A concrete example is the Lex-Opt algorithm which outputs the lexicographically first optimal allocation; the lexicographic order is with respect to the loads $(w_1, \ldots, w_n)$ of the machines.

**Theorem 6.1.** [3] **Lex-Opt is monotone**

**Proof.** Let $w = (w_1, \ldots, w_n)$ be the workload vector computed by Lex-Opt on input $s = (s_1, \ldots, s_n)$. We consider the case when machine $i$ reports a slower speed $s'_i < s_i$. Let $w'$ be the new schedule for input $s' = (s'_1, s_{-i})$. To show that Lex-Opt is monotone we need to show that $w'_i \leq w_i$.

Clearly the optimal makespan of the new speed vector can only increase, i.e. $C(w', s') \geq C(w, s)$. Let’s consider first the case where $C(w, s) = C(w, s')$. The
workload vector \( w \) is the lexicographically first and therefore \( \text{Lux-Orr} \) will select this for speeds \( s' \). Clearly in this case, \( w'_i = w_i \). In the other case, when the makespan \( C(w, s') \) is greater than \( C(w, s) \), let’s assume that machine \( i \) is the bottleneck in schedule \( w \), i.e., \( C(w, s') = \frac{w_i}{s_i} > C(w, s) \). But since \( w' \) is the lexicographically-first optimal workload for \( s' \), we have that \( C(w', s') \leq C(w, s') \), and therefore \( \frac{w_i}{s_i} \leq C(w', s') \leq C(w, s') = \frac{w_i}{s_i} \). Again, \( w'_i \leq w_i \).

We next consider randomized approximation polynomial-time mechanisms and then deterministic ones.

### 6.1 Randomized mechanisms

We will discuss 2 mechanisms in this section. The first mechanism is due to Archer and Tardos [3] and has approximation ratio 3. Later Archer [2] improved the randomized rounding procedure obtaining a 2-approximate mechanism. The second mechanism is due to Dhangwatnotai et al. [9] and is a randomized PTAS. Both mechanisms are truthful-in-expectation and they have similar approach: they create first a monotone fractional solution and then apply a randomized rounding procedure. The randomization is useful only to guarantee truthfulness and has no implication on the approximation ratio.

**A 2-approximate truthful in expectation mechanism** [3, 2]

Given the speed vector \( s = (s_1, \ldots, s_n) \), the algorithm first computes the following threshold

\[
T_{LB} = \max_j \min_i \left\{ \frac{p_j}{s_i} \sum_{k=1}^j \frac{p_k}{\sum_{l=1}^k s_l} \right\},
\]

which is a lower bound of the optimal makespan \( C(w, s) \).

Given \( T_{LB} \), the algorithm computes a fractional assignment as follows: It assigns the jobs in non-increasing order with respect to their size, i.e. \( p_1 \geq \ldots \geq p_m \). It first assigns as many jobs as possible to the fastest machine so that its load becomes equal to \( T_{LB} \). It may have to assign a fraction of some job to achieve this (thus the assignment is fractional). It continues the same procedure for the remaining machines. The threshold is such that all jobs will be assigned to the machines.

We now describe a randomized rounding procedure turn the fractional allocation into an integral one: Pick a random number \( \alpha \) uniformly at random in [0, 1]. Assume that task \( j \) is fractionally assigned to machines \( i \) and \( i+1 \). If \( x_{ij} \geq \alpha \) then assign the task to machine \( i \), otherwise assign it to machine \( j \).

**Theorem 6.2.** The above algorithm is monotone.
The Bulletin of the EATCS

Proof. All we need to prove is that the fractional workloads are monotone. This is because the expected workload of every machine is equal to the fractional workload (since \( \alpha \) was chosen uniformly).

Assume now that a machine \( i \) reports a smaller speed \( s'_i < s_i \) and let \( w \) and \( w' \) be the workload vectors. To show monotonicity we need to show \( w'_i \leq w_i \). Let \( s_i = \beta \cdot s'_i \), for some \( \beta > 1 \). The new threshold \( T'_{LB} \) can only increase, but it can be bounded by \( T'_{LB} \leq \beta \cdot T_{LB} \). If machine \( i \) had load \( T_{LB} \), then \( w'_i \leq T'_{LB} \cdot s'_i \leq \beta \cdot T_{LB} \cdot s'_i \leq T_{LB} \cdot s_i = w_i \).

If machine \( i \) was not full (that is, it had load less than \( T_{LB} \)), then it can at most take the load that exceeded the previous machines. Now that the threshold \( T'_{LB} \) has increased, the total workload on those machines with \( k < i \) can only increase. Therefore the workload of machine \( i \) can only decrease.

A PTAS truthful in expectation mechanism Very recently, Dhangwatnotai et al. [9] suggested the following randomized PTAS, that is truthful in expectation.

The algorithm first groups the jobs of size that differ within a factor of \( 1 + \epsilon \) from each other, for some small \( \epsilon \). Then it smooths the jobs, i.e., it pretends that every job has a size equal to the average of its group. Then the algorithm constructs a set \( \mathcal{P} \) of allowable (fractional) partitions of jobs to the machines, giving also a total ordering of these partitions. Then the algorithm optimizes over the partitions in \( \mathcal{P} \). From the fractional partition \( P \), we get a fractional schedule \( w(P) \), by giving to machine with the \( i \)-th slowest speed, the \( i \)-th smallest partition set. Then using randomized rounding we get an integral schedule. Finally we replace the smoothed jobs with the real ones. The algorithm does this by random shuffling.

Theorem 6.3. The above randomized PTAS is monotone.

Proof. Again it is enough to show that the fractional schedule of the smoothed jobs is monotone.

Assume that machine \( i \) reports a smaller speed \( s'_i < s_i \) and let \( s = (s_1, s_2, \ldots) \) and \( s' = (s'_1, s'_2, \ldots) \) be the corresponding speed vectors. Let machine \( i \) be the \( k \)-th slowest in \( s \) and the \( k' \)-th slowest in \( s' \), with \( k' \leq k \). Let us denote by \( P = P(s) \) and \( P' = P(s') \) the corresponding partitions chosen by the algorithm in both cases. Let us also denote by \( w(P) = (w_1(P), \ldots, w_n(P)) \), the sorted (in increasing order) workload vector with respect to the partition \( P \). We need to prove that \( w_{k'}(P') \leq w_k(P) \).

Clearly the makespans satisfy \( C(w(P'), s') \geq C(w(P), s) \), since machine \( i \) has decreased its speed. Let us first assume that the schedule induced by the partition \( P \), does not increase the makespan for \( s' \), i.e. \( C(w(P), s') = C(w(P), s) \). Therefore \( P' = P \) and since the player \( i \) has decreased its position in the sorted speed vector \( s' \), it is \( w_k(P') = w_k(P) \leq w_k(P) \).
If the schedule induced by \( P \) causes an increase of the makespan for \( s' \), then the bottleneck is one of the machines in the positions between \( k' \) and \( k \), say the machine with index \( l \in [k', k] \).

The workload of the machine in this position decreases, i.e. \( w_l(P') \leq w_l(P) \) because
\[
\frac{w_l(P)}{s_l'} \leq C(w(P'), s') \leq C(w(P), s') = \frac{w_l(P)}{s_l'}.
\]
Finally we get
\[
w_{k'}(P') \leq w_l(P') \leq w_l(P) \leq w_k(P),
\]
as needed.

### 6.2 Deterministic mechanisms

We now consider deterministic mechanisms for the related machines problem. We distinguish 2 cases: the case of fixed number of machines (for which there is a FPTAS for the non-mechanism-design version) and the case of variable number of machines (for which there is a PTAS but no FPTAS unless \( P=NP \)).

#### 6.2.1 Fixed number of machines

Auletta et al. [4] give the first deterministic polynomial-time monotone algorithm for the fixed number of machines problem. Their algorithm is 4-approximate. The algorithm schedules optimally the \( h \) largest jobs, for some parameter \( h \) and it assigns the rest of the jobs in a greedy fashion. A central point of their approach is that the greedy allocation is monotone for the special case when the speeds are powers of 2. They first round down the originals speeds in the closest power of 2, and then apply their monotone algorithm.

This result was improved by Andelman et al. [1] who gave a PTAS and a different mechanism FPTAS. The PTAS algorithm first modifies the set \( M \) of the jobs to a set \( M' \) as follows: It partitions the jobs into a set \( B \) of big jobs, and a set \( S \) of small jobs. A job is in \( B \) if its size is above some threshold \( T \). Then, jobs in \( S \) are packed into chunks of size in \([T/2, T]\) (the last chunk may have size than \( T/2 \)). Let us call the set of chunks \( S' \). The working set of jobs is the merge of the two sets, \( M' = B \cup S' \), for which we find the optimal assignment applying the lexicographically first optimal algorithm \( \text{Lex-Opt} \).

The algorithm is trivially monotone, as the construction of the modified job set \( M' \) is independent of the speed vector and because the \( \text{Lex-Opt} \) algorithm is monotone, as we showed in Theorem 6.1.

Andelman et al. [1] gave a different monotone FPTAS for the problem. The algorithm takes any black-box algorithm with approximation ratio \( c \) and transforms it to a monotone algorithm with approximation \( c(1 + \epsilon) \), for every \( \epsilon > 0 \). They use this on the FPTAS of [15].

The transformation is performed in 3 steps:
1. In the first step the algorithm produces a modified vector of speeds $d$ as follows: First it rounds the speeds down to powers of $(1 + \epsilon)$. Then it normalizes the vector such that $d_n = 1$. Finally, it rounds the machines that are very slow, with respect to some threshold $L$, to $(1 + \epsilon)^{-L}$.

2. In the second step the algorithm performs an enumeration over all the different vectors $d'$ with speeds $(1 + \epsilon)^{-i}$, with $i \in \{0, L\}$. For every such vector $d'$, it applies the black-box algorithm. Finally it sorts the workloads such that the machine with the $i$-th smallest speed will get the $i$-th smallest workload.

3. In the final step, it tries all the sorted assignments to $d$ and outputs the assignment that minimizes the makespan (choosing the lexicographically first in case of ties).

6.2.2 Arbitrary number of machines

The following algorithm due to Andelman et al. [1] is based on the ideas of the algorithm of Archer and Tardos (Section 6.1) and has approximation ratio 5. To overcome the problem of derandomizing imposed by monotonicity they modify the speed set.

The currently best deterministic algorithm is due to Kovacs [18]. The algorithm first rounds the speeds down to the closest power of 2, i.e. $d_i = 2^\lceil \log_2 s_i \rceil$. Then it runs the well-known algorithm Longest Processing Time first (LPT) on the modified speed vector $d$. Finally, among machines of the same rounded speed, the algorithm reorders the assigned work such that $w_i \leq w_{i+1}$. The algorithm is monotone and attains approximation ratio 2.8 [19]. The proof of its monotonicity is complicated and it is beyond the scope of this article.

7 Conclusions

The scheduling problem with its many facets is one of the driving problems of the area of Algorithmic Mechanism Design. There are many interesting open problems, but we feel that the following are the most important:

- Characterize the set of truthful mechanisms for unrelated machines.
- Close the gap between the lower (2.61) and the upper ($n$) bound on the approximation ratio for unrelated machines. Also important are the same questions about the fractional and randomized case.
- Give a deterministic PTAS mechanism for the related machines problem or prove that none exists.
References


The Complexity of Planar Graph Isomorphism

Jacobo Torán and Fabian Wagner*

Abstract

The Graph Isomorphism problem restricted to planar graphs has been known to be solvable in polynomial time many years ago. In terms of complexity classes however, the exact complexity of the problem has been established only very recently. It was proved in [6] that planar graph isomorphism can be computed within logarithmic space. Since there is a matching hardness result [12], this shows that the problem is complete for L. Although this could be considered as a result in algorithmics its proof relies on several important new developments in the area of logarithmic space complexity classes and reflects the close connections between algorithms and complexity theory. In this column we give an overview of this result mentioning the developments that led to it.

∗Universität Ulm, Institut für Theoretische Informatik, fabian.wagner@uni-ulm.de
1 Introduction

The Graph Isomorphism problem asks whether two given graphs are isomorphic or in other words whether there is a bijection between the nodes of the two graphs, preserving the adjacency relation. Graph Isomorphism is one of the few problems in \( \text{NP} \) that is not known to be in \( \text{P} \) or \( \text{NP} \)-complete. On the other hand, for many restricted graph classes like trees, graphs of bounded degree or partial \( k \)-trees, efficient algorithms for the isomorphism problem are known. We consider in this column the class of planar graphs and for simplicity restrict ourselves to undirected simple graphs. A graph is planar if it can be drawn in the plane without any crossing edges. A special class of planar graphs is that of 3-connected planar graphs. A graph is \( k \)-connected if it remains connected after deleting \( k \) arbitrary vertices. It was shown in 1933 that 3-connected planar graphs have exactly two planar embeddings [22]. This fact was used by Weinberg in 1966 to give an \( O(n^2) \) algorithm for testing isomorphism of 3-connected planar graphs [21] (although the paper, \( n \) is the number of vertices of the input graphs). The idea of the algorithm is simple, the two given graphs are embedded in the plane and it is tested whether the embedding of the first graph is isomorphic to one of the two possible embeddings of the second graph. It suffices to fix an arbitrary edge \( e \) in one graph and run through all the edges \( e' \) of the other graph, i.e. test whether there exists an automorphism which maps \( e \) onto \( e' \). With this method it is also possible to efficiently assign codes to 3-connected planar graphs as a way to identify them. This means that there is an efficiently computable function mapping graphs to strings or codes, so that two graphs are mapped to the same code if and only if they are isomorphic. Hopcroft and Tarjan extended Weinberg’s algorithm and gave the first polynomial time algorithm for the isomorphism of planar graphs [8]. In their algorithm the graphs are divided first in connected components, these are subdivided in biconnected components and finally these components are partitioned in triconnected components. An initial connected component has articulation points separating its biconnected components from each other. These initial components are represented by a tree-like structure containing vertices for articulation points and vertices for biconnected components. In a similar way, the biconnected components are represented by tree structures containing vertices for its triconnected components and vertices for pairs of nodes (called separating pairs) which are separating these components. A triconnected component is either a 3-connected graph, a cycle or a 3-bond. Each vertex representing a triconnected component is then labeled with the code of the component and the whole structure can then be tested for isomorphism in a similar way as it is done for tree isomorphism. The original algorithm had a running time of \( O(n^2) \). This was then improved to \( O(n \log n) \) in [10] and finally Hopcroft and Wang [11] obtained a linear time algorithm for the isomorphism of planar graphs. Recently Kukluk, Holder, and
Cook [13] gave an $O(n^2)$ algorithm for planar graph isomorphism, which is suitable for practical applications.

Regarding the parallel complexity of the problem, Miller and Reif [16] gave the first NC algorithm for planar graph isomorphism. Their algorithm runs in time $O(\log n)$ with a polynomial number of processors in the CRCW PRAM model. This corresponds to the complexity class $\text{AC}^1$ of problems computable by unbounded fan-in circuits of polynomial size and logarithmic depth. More recently, using a completely different approach based on descriptive complexity, Grohe and Verbitsky [7] provided a new method for testing isomorphism of planar graphs also within $\text{AC}^1$. They proved that for a class $\mathcal{G}$ of graphs, if every graph in the class is definable in first order logic with a finite number of variables and logarithmic quantifier depth, then the isomorphism problem for $\mathcal{G}$ is in $\text{AC}^1$. Verbitsky [20] showed that the class of planar 3-connected graphs is definable with 15 variables and logarithmic quantifier depth. Together with the $\text{AC}^1$ reduction from planar isomorphism to 3-connected planar isomorphism from [16] this provides a different way to show that planar isomorphism lies in $\text{AC}^1$.

We describe in this survey some of the results leading to the improvement of the upper bound from $\text{AC}^1$ to logarithmic space. Located between $L$ and $\text{AC}^1$, the complexity class $\text{UL}$ played an important part in the development of the results. $\text{UL}$ (unambiguous logarithmic space) is the class of problems computable by a logarithmic space nondeterministic machine having at most one accepting computation path for each input. The relation between the considered complexity classes is as follows:

$$L \subseteq \text{UL} \subseteq \text{AC}^1.$$

We denote by $\text{FL}$ the class of functions computable within logarithmic space.

All the isomorphism results described in this overview can in fact be extended to graph canonization results. For a class $\mathcal{G}$ of graphs we say that a function $f : \mathcal{G} \to \{0, 1\}^*$ computes a complete invariant for the class if for every $G, H \in \mathcal{G}$ it holds that $G$ and $H$ are isomorphic if and only if $f(G) = f(H)$. If moreover $f$ computes for each $G$ a graph $f(G)$ isomorphic to $G$ then we call $f$ a canonizing function and $f(G)$ a canon.

We recall in Section 2 several facts that were used in the proof of the results. Similar as in the developments leading to polynomial time test for planar graph isomorphism, the first logarithmic space isomorphism algorithms worked for trees and 3-connected graphs. These are overviewed in Sections 2 and 3. Finally the logarithmic space algorithm for planar graph isomorphism is explained in Section 4.
2 Some previous results

2.1 Tree isomorphism in \( L \)

Lindell gave in [14] a logarithmic space algorithm for tree isomorphism and can-
onization. Some of the ideas in this algorithm are used also in the new results. Lindell defined a canonical ordering between trees. For a tree \( T \), we represent its
root by \( t \) and the out-degree of \( t \) by \( \#t \). The size of the tree is denoted \( |T| \). The tree
isomorphism ordering \( <_T \) from Lindell is defined in the following way: Given two
trees \( S \) and \( T \), we say \( S <_T T \) if:

\begin{itemize}
  \item \( |S| < |T| \) or
  \item \( |S| = |T| \) and \( \#s < \#t \), or
  \item \( |S| = |T| \), \( \#s = \#t = k \) and \( (S_1, \ldots, S_k) < (T_1, \ldots, T_k) \) lexicographically,
    were \( S_1 \leq_T S_2 \leq_T \ldots \leq_T S_k \), and \( T_1 \leq_T T_2 \leq_T \ldots \leq_T T_k \), are the ordered
    subtrees of \( S \) and \( T \) rooted at the children of \( s \) and \( t \).
\end{itemize}

It is not hard to see that if neither \( S <_T T \) nor \( T <_S S \) then \( S \) and \( T \) are
isomorphic. Obviously the first two tests in the definition of tree ordering can be
done in logarithmic space. Lindell proved that this is also true for the third step. This is done with the following variant of depth first search:

- Partition the children into classes, according to their size. Find the number
  of children of \( s \) and \( t \) of minimal size. If these numbers do not coincide
  then declare the tree with the largest number of minimal size children to be
  the smaller one. Otherwise, check the next size class until an inequality is
  found or all the children have been considered.

- If \( s \) and \( t \) have the same number of children of each size, then we compare
  the children in each size class in increasing order of the sizes recursively
  as follows: Let \( k \) be the number of children in one size class. We can
  suppose \( k > 0 \).

  - If \( k = 1 \) then only one recursive call is made and no extra space is needed
    for this.

  - If \( k > 1 \) then for each node in the size class in \( S \) we compute its order
    profile. This consists of three counters \( c_<, c_> \) and \( c_= \) indicating the
    number of children in the corresponding size class of \( T \) being respectively
    smaller than, larger than or equal to the node under consideration. The
    counters are updated by making cross comparisons. We start with the
children with minimal order profile, those with \( c_r = 0 \). They form an isomorphism class. The size of this class is compared across the trees by comparing the values of the \( c_r \) counters. If they match, both trees have the same number of minimal children. To compare larger children in the same size class, the value of \( c_r \) in the last step works as a threshold. This is used to search for the next minimal children of \( s \) and \( t \). The process is then repeated and the threshold is incremented until reaching \( k \), at which point we proceed to the next size class. If all the size classes are visited without detecting an inequality then the trees are isomorphic.

### 2.2 Planarity testing and distance computation

A graph is planar if it can be drawn on the plane so that no edges cross. Such a drawing is a planar embedding. Allender and Mahajan [3] showed that the problem of determining if a given graph is planar, can be computed in the complexity class \( SL \) (symmetric logarithmic space). Some year later Reingold [18] proved that the classes \( SL \) and \( L \) coincide, thus bringing the planarity recognition problem to \( L \).

Let \( E_v \) be the set of edges \((u, v)\) in the graph \( G = (V, E) \) incident to \( v \). A rotation scheme for a graph \( G \) is a set \( \rho \) of permutations, \( \rho = \{ \rho_v | v \in V \} \), where \( \rho_v \) is a permutation on \( E_v \) that has only one cycle (which is called a rotation). Let \( \rho^{-1} \) be the set of inverse rotations, \( \rho^{-1} = \{ \rho^{-1}_v | v \in V \} \). A rotation scheme \( \rho \) describes an embedding of graph \( G \) in the plane. If the embedding is planar, we call \( \rho \) a planar rotation scheme. The pair \((G, \rho)\) is called a combinatorial embedding for \( G \). Allender and Mahajan also showed the following useful result:

**Theorem 2.1.** [3] There is a logarithmic space algorithm that on input a planar graph \( G \) produces a planar rotation scheme \( \rho \) for \( G \).

A planar 3-connected graph has exactly two planar rotation schemes [22], some rotation scheme \( \rho \) and its inverse \( \rho^{-1} \).

An important tool in the first isomorphism tests for 3-connected planar graphs is the computation of distances in planar graphs within the class \( UL \). This theorem builds on a series of results dealing with the reachability problem in directed planar graphs, [2],[17] that lead to the algorithm from Bourke, Tewari and Vinodchandran [4] to compute the reachability problem for planar graphs within \( UL \cap coUL \).

**Theorem 2.2.** [19] The distance between two given vertices in a planar graph can be computed in \( UL \cap coUL \).
2.3 Universal exploration sequences

The celebrated result from Reingold [18] showing that the reachability problem in undirected graphs can be computed in \(L\), has an important consequence for the construction of universal exploration sequences in logarithmic space. This fact is used in some of the isomorphism algorithms.

For a \(d\)-regular graph \(G\), a numbering of the edges and a starting edge \(e_0\), a sequence \((\tau_1, \tau_2, \ldots, \tau_l) \in \{0, \ldots, d - 1\}^l\) defines a walk \(v_{-1}, v_0, \ldots, v_k\) in \(G\) in the following way: starting at \(e_0 = (v_{-1}, v_0)\), for each \(i\) if \((v_{i-1}, v_i)\) is the \(k\)-th edge of \(v_i\) then \((v_i, v_{i+1})\) is the \(k + \tau_i\)-th edge of \(v_i\) modulo \(d\).

A sequence \((\tau_1, \tau_2, \ldots, \tau_l) \in \{0, \ldots, d - 1\}^l\) is called an \((n, d)\) universal exploration sequence if for every connected \(d\)-regular graph with at most \(n\) vertices, any numbering of its edges and any starting edge, the walk obtained from the sequence explores all the vertices in the graph.

We use the fact that such universal exploration sequences exist and can be computed in logarithmic space.

**Theorem 2.3.** [18] There exists a logarithmic space algorithm that on input \((1^n, 1^d)\) produces an \((n, d)\)-universal exploration sequence of polynomial size.

3 Planar 3-connected Graph Isomorphism

Weinberg’s [21] \(O(n^2)\) algorithm for testing isomorphism of planar 3-connected graphs constructs for a graph \(G\) a code with respect to each one of its edges and both rotation schemes. Of all these codes, the lexicographically smallest one is used as a canonical form for \(G\). Weinberg’s algorithm does not work within logspace, because the vertices and edges have to be stored. Thierauf and Wagner showed in [19] how to construct a different code in \(UL\). Some months later, using Reingolds results on logarithmic space universal exploration sequences [18], Datta, Limaye and Nimbhorkar improved this to an isomorphism algorithm for planar 3-connected graphs that works in logarithmic space [5]. We describe both results in this section.

3.1 An isomorphism algorithm in \(UL \cap coUL\)

**Theorem 3.1.** [19] The isomorphism problem for planar, 3-connected graphs is in \(UL \cap coUL\).

Let \((s, t)\) be a designated edge and \(\rho\) be a rotation scheme for \(G\). The construction has three steps: First, we compute a canonical spanning tree \(T\) for \(G\). Second, with help of this spanning tree and the rotation function \(\rho\) we perform a
depth-first traversal on the edges of the graph and construct a canonical list \( L \) of all edges of \( G \). Finally, we rename the vertices depending on the position of their first occurrence in the list \( L \).

We will see that the spanning tree in step 1 can be computed in (the functional version of) \( \text{UL} \cap \text{coUL} \). The list and the renaming in step 2 and step 3 can be computed in \( \text{FL} \).

The overall algorithm has to decide whether two given graphs \( G \) and \( H \) are isomorphic. To do so, we fix \((s, t)\) and \( \rho \) for \( G \) and cycle through all edges of \( H \) as designated edge and the two possible embeddings of \( H \). Then \( G \) and \( H \) are isomorphic if and only if the canonical forms for \( G \) and \( H \) match. It is not hard to see that this outer loop works in logspace.

**Step 1: Construction of a canonical spanning tree**

We show that the following problem can be solved in unambiguous logspace. Given, an undirected graph \( G = (V, E) \), a rotation scheme \( \rho \) for \( G \), and a designated edge \((s, t) \in E\), output a canonical spanning tree \( T \subseteq E \) of \( G \), which does not depend on the input representation of \( \rho \) for \( G \), any representation will result in the same spanning tree \( T \).

The idea to construct the spanning tree is to traverse \( G \) with a breadth-first search starting at node \( s \). The neighbors of a node are visited in the order given by the rotation scheme \( \rho \). Since the algorithm should work in logspace, we cannot afford to store all the nodes that we already visited, as in a standard breadth-first search. We get around this problem by working with distances between nodes.

We start with the nodes at distance 1 from \( s \). That is, write \((s, v)\) on the output tape, for all \( v \in \Gamma(s) \). Now let \( d \geq 2 \) and assume that we have already constructed \( T \) up to nodes at distance \( \leq d - 1 \) from \( s \). Then we consider the nodes at distance \( d \) from \( s \). Let \( w \) be a node with distance \( d(s, w) = d \). We connect \( w \) to the tree constructed so far by computing a shortest path from \( s \) to \( w \). Ambiguities are resolved by using the first feasible edge according to \( \rho \). We start with \((s, t)\) as the active edge \((u, v)\).

- If \( d(u, w) > d(v, w) \), then \((u, v)\) is the first edge encountered that is on a shortest path from \( u \) to \( w \). Therefore we go from \( u \) to \( v \) and start searching the next edge from \( v \). As starting edge we take \( \rho(v, u) \), the successor of \((v, u)\). This is the new active edge.
- If \( d(u, w) \leq d(v, w) \), then \((u, v)\) is not on a shortest path from \( u \) to \( w \). Then we proceed with \( \rho(v, u) \) as the new active edge.

After \( d - 1 \) steps in direction of \( w \), the node \( v \) of the active edge \((u, v)\) is a predecessor of \( w \) on a shortest path from \( s \) to \( w \). Then we write \((v, w)\) on the output tape.
The spanning tree $T$ is canonical, because its construction depends only on $\rho$, edge $(s, t)$, and edge set $E$. The following figure shows an example of a spanning tree $T$ for a graph $G$ with rotation function $\rho$ which arranges the edges in clockwise order around each vertex of $G$.

$$\rho = \{\rho_s, \rho_t, \rho_{v_1}, \rho_{v_2}, \rho_{v_3}\}$$

$$\rho_s = ( (s, t) (s, v_1) (s, v_2) )$$

$$\rho_t = ( (t, s) (t, v_1) (t, v_1) )$$

$$\rho_{v_1} = ( (v_1, s) (v_1, t) (v_1, v_3) (v_1, v_2) )$$

$$\rho_{v_2} = ( (v_2, s) (v_2, v_1) (v_2, v_1) )$$

$$\rho_{v_3} = ( (v_3, t) (v_3, v_2) (v_3, v_1) )$$

Figure 1: A 3-connected planar graph and its planar rotation scheme.

Except for the computation of the distances, the algorithm works in logspace. We have to store the values of $d$, $k$, $u$ and $v$, and the position of $w$, plus some extra space for doing calculations. Thierauf and Wagner also proved, that distances in planar graphs can be computed in $UL \cap coUL$. This is based on the results from Reinhard and Allender [17] and Bourke, Tewari, and Vinodchandran [4], that the reachability problem for planar directed graphs is in $UL \cap coUL$. Since $L_{UL \cap coUL} = UL \cap coUL$ the canonical spanning tree can be computed in $UL \cap coUL$.

**Step 2: Computation of a canonical list of all edges**

With $G = (V, E)$, a rotation scheme $\rho$ for $G$, a spanning tree $T \subseteq E$ of $G$ and a designated edge $(s, t) \in T$ we compute a canonical list $L$ of all edges in $E$. The list $L$ then still contains the original vertex names in $G$, it does not depend otherwise on the input representation of $\rho$, $G$ or $T$.

The idea is to traverse the spanning tree in a depth-first manner. At each vertex $u$ we visit all incident edges of $u$ in a cyclic manner according to $\rho_u$ until the next edge $e$ of the spanning tree is reached. We go down the tree along $e$ and recursively do the same at the node reached. At some point we will encounter $e$ again and come back to $u$. Then we continue to output the edges incident to $u$.

More formally, we start the traversal with edge $(s, t)$ as the active edge $(u, v)$. We write $(u, v)$ on the output tape and then compute the next active edge as follows:

- If $(u, v) \in T$ then we walk depth-first in $T$ from $u$ to $v$, consider the edge $(v, u)$ and take $\rho_v(v, u)$ its successor according to $\rho_v$.
• If $(u, v) \notin T$ then we proceed breadth-first with $\rho_u(u, v)$.

This step is repeated until we entirely traversed $E$ and the active edge is again $(s, t)$. 
Every undirected edge is encountered exactly once in each direction. According to Figure 1 the canonical list $L$ is the following.

\[
L = (s, t)(t, v_3)(v_3, v_2)(v_1, v_1)(v_2)(t, s) \\
(s, v_1)(v_1, v_2)(v_2)(v_1, v_1)(v_1, s) \\
(s, v_2)(v_2, v_2)(v_2)(v_1, s)
\]

**Step 3: Renaming the vertices**

The last step is to rename the vertices in the list $L$ so that they become independent of the names they have in $G$. This is achieved as follows: consider the first occurrence (from left) of node $v$ in $L$. Let $k - 1$ be the number of pairwise different nodes to the left of $v$. Then all occurrences of $v$ are replaced by $k$. Recall, that $L$ starts with the edge $(s, t)$. Hence, all occurrences of $s$ get replaced by 1, all occurrences of $t$ by 2, and so on. Call the new list $\text{code}(G, \rho, s, t)$.

Given $L$ as input, the list $\text{code}(G, \rho, s, t)$ can be computed in logspace. We start with the first node $v$ (which is $s$) and a counter $k$, that counts the number of different nodes we have seen so far. In the beginning, we set $k = 1$.

• If $v$ occurs for the first time, then we output $k$ and increase $k$ by 1.

• If $v$ occurs already to the left of the current position then we have to determine the number, that $v$ got at its first occurrence. To do so, we determine the first appearance of $v$ and then count the number of different nodes to the left of $v$ at its first appearance.

Then we go to the next node in $L$.

Consider the example from above. The code constructed from list $L$ for $G$ is as follows.

\[
L = (s, t)(t, v_3)(v_3, v_2)(v_1, v_1)(v_2)(t, s) \\
\text{code}(G, \rho, s, t) = (1, 2)(2, 3)(3, 4)(3, 5)(3, 2)(2, 5)(2, 1) \\
\text{sequel of } L = (s, v_1)(v_1, t)(v_1, v_2)(v_1, v_1)(v_1, v_1)(v_1, s) \\
\text{sequel of } \text{code} = (1, 5)(5, 2)(5, 3)(5, 4)(5, 1) \\
\text{sequel of } L = (s, v_2)(v_2, v_2)(v_2, v_3)(v_2, s) \\
\text{sequel of } \text{code} = (1, 4)(4, 5)(4, 3)(4, 1)
\]

It remains to argue that the new names of the nodes are independent of their names in $G$. Let $H$ be a graph which is isomorphic to $G$, and let $\varphi$ be an isomorphism between $G$ and $H$. Note that $\rho \circ \varphi$ is a rotation scheme for $H$. Consider
The Bulletin of the EATCS

the computation of the code for graph $H$ with rotation scheme $\rho \circ \varphi$ and designated edge $(\varphi(s), \varphi(t))$. The spanning tree computed in step 1 will be $\varphi(T)$ and the list computed in step 2 will be $\varphi(L)$. Now the above renaming procedure will give the same number to node $v$ in $L$ and to node $\varphi(v)$ in $\varphi(L)$. For example, the nodes $\varphi(s)$ and $\varphi(t)$ will get number 1 and 2, respectively. It follows that $\text{code}(G, \rho, s, t) = \text{code}(H, \rho \circ \varphi, \varphi(s), \varphi(t))$. We summarize:

**Theorem 3.2.** [19] Let $G$ and $H$ be connected, undirected graphs, let $\rho_G$ be a rotation scheme for $G$ and $(s, t)$ be an edge in $G$. Then $G$ and $H$ are isomorphic if there exists a rotation scheme $\rho_H$ for $H$ and an edge $(u, v)$ in $H$ such that $\text{code}(G, \rho_G, s, t) = \text{code}(H, \rho_H, u, v)$.

With a very different approach, Datta, Limaye and Nimbhorkar [5] improved the previous result from $UL \cap \text{coUL}$ to $L$. Their method is in some sense much easier since it avoids the spanning tree construction eliminating the distance computations (the part in $UL \cap \text{coUL}$). It uses however the concept of universal exploration sequence and the non-trivial fact that such sequences can be computed in $L$.

**Theorem 3.3.** [5] The isomorphism problem for planar, 3-connected graphs is in $L$.

The idea of the algorithm is to use a universal sequence [18] in order to construct a canonical code for a given planar 3-connected graph $G$. Since Reingold’s construction requires the graph to have constant degree, there is a preprocessing step in which $G$ is transformed into a 3-regular colored graph $G'$ with the property that two graphs are isomorphic if and only if their transformations are also isomorphic (with a color preserving isomorphism). In a second step a canonical code is computed. The code is specific to the choice of a planar embedding $\rho$ for $G$, a starting node and a starting edge. Since there are only polynomially many possible choices for these parameters, for two given graphs $G$ and $H$, a logarithmic space procedure can cycle through all the possibilities and decide that the graphs are isomorphic if and only if the canonical codes match for any of the choices.

**Step 1: Making the graph 3-regular**

Given a 3-connected planar graph $G = (V, E)$ and a planar embedding $\rho$ we construct a 3-regular planar graph $G'$ with the edges colored with two colors. $G'$ might not be 3-connected, however the planar embedding from $G$ will be inherited to $G'$. Every vertex $v$ of $G$ is replaced in $G'$ by a cycle $[v_1, \ldots, v_d]$ ($d$ is the degree of $v$). The $d$ edges $e_1, \ldots, e_d$ incident with $v$ in $G$ are now respectively incident to $[v_1, \ldots, v_d]$ in $G'$. The edges of the cycles are colored with color 1.
and the edges $e_1, \ldots, e_d$ with color 2. The obtained graph $G'$ is 3-regular and it is not hard to see that two graphs $G$ and $H$ are isomorphic if and only if their transformation $G'$ and $H'$ are isomorphic with a color preserving isomorphism.

**Step 2: Obtaining the canonical code**

On input an edge-colored graph $G$ with $n$ vertices, maximum degree 3, a planar embedding $\rho$, a starting vertex $v$ and a starting edge $e = (u, v)$, we want to construct a canon for $G$. For this, we compute first in logarithmic space an $(n, 3)$-universal exploration sequence $U$. Then, starting at $v$ and $e$ we transverse $G$ according to $U$ and $\rho$ giving the list $L$ of the visited vertices as label. We can rename the vertices according to their first occurrence in $L$, as it is done in step 3 from Theorem 3.1. Finally, we can cycle over every possible pair $(i, j)$, checking whether it is an edge in the renamed list and outputting its color if this is the case. This output can be considered as a canonical colored adjacency matrix for $G$.

The authors prove that this method is correct by showing the following: For two graphs $G_1$ and $G_2$ with their respective embedding $\rho_1, \rho_2$, with starting vertices $v_1, v_2$ and edges $e_1, e_2$, if the canons coincide then the graphs are isomorphic, and moreover, if $G_1$ is isomorphic to $G_2$, then there is some choice of the parameters that makes their respective canons equal.

## 4 Planar Graph Isomorphism

In this section we describe the log-space algorithm for planar graph isomorphism. A previous step towards this result was a logarithmic space isomorphism test for partial 2-trees [1]. Partial 2-trees are a subclass of the planar graphs coinciding with that of series-parallel graphs and contain all outer-planar graphs. For proving this result Arvind, Das and Köbler represent a partial 2-tree as a tree of cycles. Similar to Lindell’s algorithm [14] they compare two such tree representations up to isomorphism, defining a canonical ordering procedure, which finally gives a canonization algorithm.

In the isomorphism algorithm for general planar graphs a similar representation is used, namely a tree of triconnected components. A triconnected component is a 3-connected graph, a cycle or a 3-bond, i.e. two vertices connected by three edges.

We give a log-space algorithm for the graph canonization problem for planar graphs, to which planar graph isomorphism reduces. The canonization involves assigning to each graph an isomorphism invariant string of polynomial length.

The algorithm decomposes first the planar graph into its biconnected components and constructs a biconnected component tree in log-space [1]. Then, it
The Bulletin of the EATCS

further decomposes the biconnected planar components into their triconnected components to obtain a triconnected component tree in log-space. Hopcroft and Tarjan [9] presented a sequential algorithm for the decomposition of a biconnected planar graph into its triconnected components. This method can be adapted to work in log-space. The algorithm recursively removes separating pairs from the graph and puts a copy of the separating pair in each of the split components so formed, i.e. the nodes in the separating pair are connected by a virtual edge. The decomposition stops when the split components become triconnected. Define a node for each separating pair and each component and connect a separating pair node with a triconnected component node if the separating pair is contained in the component. The resulting graph is a tree, the triconnected component tree. This decomposition is unique [15]. Datta et. al. [6] prove, that such a decomposition can be computed in log-space. Figure 2 shows an example of the decomposition of a biconnected planar graph $G$. Its triconnected components are $G_1, \ldots, G_4$ and the corresponding triconnected component tree is $T$. In $G$, the pairs $(a, b)$ and $(c, d)$ are the separating pairs. Since the 3-connected separating pair $(c, d)$ is connected by an edge in $G$, we also get $\{c, d\}$ as triple-bond $G_3$. The virtual edges corresponding to the separating pairs are drawn with dashed lines.

![Diagram](image_url)

**Figure 2:** Decomposition of a biconnected planar graph into a triconnected component tree.

The triconnected components can be canonized in log-space [5]. Hence, for triconnected component trees, compute their canonical invariant in log-space, i.e. two biconnected graphs are isomorphic if their trees are found to be equal.

In section 4.1, we summarize, how to canonize biconnected planar graphs by applying tree canonization ideas from [14] to their triconnected component trees. Note, that pairwise isomorphism of two trees labelled with the canons of their components does not imply isomorphism of the corresponding graphs. Lindell’s algorithm and complexity analysis had to be modified in a non-trivial way for this step to work in log-space.

In section 4.2, we describe, how to canonize planar graphs using their bicon-
connected component trees, again using the basic structure of Lindell’s algorithm. The comparison algorithm refers to the biconnected component tree of the planar graph and when comparing biconnected components, to their triconnected component trees. This requires a detailed analysis of the interferences of both tree structures.

### 4.1 Canonization of biconnected planar graphs

Let $S$ and $T$ be two triconnected component trees for the biconnected planar graphs $G$ and $H$, respectively. $S$ and $T$ are rooted at separating pair nodes, say $s = (a, b)$ and $t = (a', b')$. Therefore we also write $S_{(a, b)}$ and $T_{(a', b')}$. They have separating pair nodes at odd levels and triconnected component nodes at even levels. Figure 3 shows two trees to be compared.

![Figure 3: Triconnected component trees.](image)

Similar as in Lindell’s algorithm, we define the isomorphism order of two triconnected component trees $S$ and $T$ rooted at separating pairs $s = (a, b)$ and $t = (a', b')$. $S_{(a, b)} <_T T_{(a', b')}$ if:

1. $|S_{(a, b)}| < |T_{(a', b')}|$ or
2. $|S_{(a, b)}| = |T_{(a', b')}|$ but $s < t$ or
3. $|S_{(a, b)}| = |T_{(a', b')}|$, $s = t = k$, but $(S_{G_1}, \ldots, S_{G_k}) <_T (T_{H_1}, \ldots, T_{H_k})$ lexicographically, where we assume that $S_{G_1} \leq_T \cdots \leq_T S_{G_k}$ and $T_{H_1} \leq_T \cdots \leq_T T_{H_k}$ are the ordered subtrees of $S_{(a, b)}$ and $T_{(a', b')}$, respectively. To compute the order between the subtrees $S_{G_i}$ and $T_{H_i}$ we compare lexicographically the canons of $G_i$ and $H_i$ and recursively the subtrees rooted at the children of $G_i$ and $H_i$. Note that these children are again separating pair nodes.

72
4. \(|S_{(a,b)}| = |T_{(a',b')}|, \#s = \#t = k, (S_{G_i} \leq_T \ldots \leq_T S_{G_k}) = (T_{H_i} \leq_T \ldots \leq_T T_{H_k})\),
but \((O_1, \ldots, O_p) < (O_1', \ldots, O_p')\) lexicographically, where \(O_j\) and \(O_j'\) are the orientation counters of the \(j\)th isomorphism classes \(I_j\) and \(I'_j\) of all the \(S_{G_i}'s and the \(T_{H_i}'s\). (The concept of orientation counter is explained later).

We say that two triconnected component trees \(S_e\) and \(T_e'\) are equal according to the isomorphism order, denoted by \(S_e \equiv_T T_e'\), if neither \(S_e < T_e'\) nor \(T_e' < S_e\) holds. Two trees are \(\equiv_T\)-equal, precisely when the underlying graphs are isomorphic.

We summarize now, how we can compute the isomorphism order when we compare subtrees rooted at separating pairs, e.g. \(S_{(a,b)}\) and \(T_{(a',b')}\), and when we compare subtrees rooted at triconnected components, e.g. \(S_{G_i}\) and \(T_{H_j}\).

Comparing \(S_{(a,b)}\) and \(T_{(a',b')}\) is similar to the comparison of subtrees in Lindell’s algorithm. We make a cross-comparison of the children and store the counters \(c_c, c_w, c_z\) for their order profile.

Assume, both subtrees are of equal size, i.e. \(|S_{G_i}| = |T_{H_j}| = N\), both rooted at triconnected component nodes \(G_i\) and \(H_j\), respectively.

First, we compare the types of \(G_i\) and \(H_j\). We say that bonds \(\leq_T\) cycles and cycles \(\leq_T\) 3-connected components. 3-bonds are always equal. If both are cycles or 3-connected components then we construct the canons of \(G_i\) and \(H_j\) and compare all of them bit-by-bit.

To canonize a cycle, we traverse it starting from the virtual edge which corresponds to its parent (i.e. the parent node of \(G_i\)), and then traversing the entire cycle along the edges encountered. There are two possible traversals depending on which direction of the starting edge is chosen. Thus, a cycle has two possible canons.

To canonize a 3-connected component \(G_i\), we use the log-space algorithm from Datta, Limaye, and Nimbhorkar [5]. The canon depends on the direction of the starting edge and additionally, on the embedding of the component \(G_i\). For 3-connected components, there are two possible embeddings. Hence, we have up to four possible canons.

In the bit-by-bit comparison, we have to distinguish several cases. When we reach virtual edges in the comparison steps, we go into recursion at the subtrees rooted at the corresponding separating pairs. If we find in the recursion that one of the subtrees is smaller than the other, then we have found an inequality between the current canons we compare. We eliminate the canons which are not found to be minimal. At the end, if there remains a canon for \(G_i\) and for \(H_j\), then both subtrees \(S_{G_i}\) and \(T_{H_j}\) are equal up to step 3.
Orientation counters. Here it does not suffice to stop after step 3. We need a further comparison step to ensure that $G$ and $H$ are indeed isomorphic. We give an example illustrating this in Figure 4. Assume that $s$ and $t$ have two children each, $G_1$, $G_2$ and $H_1$, $H_2$ such that $G_1 \cong H_1$ and $G_2 \cong H_2$. Still we cannot conclude that $G$ and $H$ are isomorphic because it is possible that the isomorphism between $G_1$ and $H_1$ maps $a$ to $a'$ and $b$ to $b'$, but the isomorphism between $G_2$ and $H_2$ maps $a$ to $b'$ and $b$ to $a'$. Then these two isomorphisms cannot be extended to an isomorphism between $G$ and $H$.

![Figure 4:](image)

To handle this problem, we introduce the notion of an orientation of a separating pair. A separating pair gets an orientation from subtrees rooted at its children. Also, every subtree rooted at a triconnected component node gives an orientation to the parent separating pair. If the orientation is consistent, then we define $S(a,b) = T(a',b')$ and we will show that $G$ and $H$ are isomorphic in this case.

We define the orientation given to the parent separating pair of $G_i$ and $H_j$ as the direction in which the minimum canon traverses this edge. If the minimum canons are obtained for both choices of directions of the edge then we say that $S_G$ and $T_H$ are symmetric about their parent separating pair, and thus do not give an orientation.

We define the orientation given to the virtual edge in the parent triconnected component of the corresponding separating pair node $(a,b)$ or $(a',b')$ by considering all the orientations given to the separating pair of their children $G_1, \ldots, G_k$, respectively. We first order the subtrees, say $S_{G_1} \leq \cdots \leq S_{G_k}$ and $T_{H_1} \leq \cdots \leq T_{H_k}$, and partition them into isomorphism classes, say $I_1, \ldots, I_p$ and $I'_1, \ldots, I'_p$. Let $I_j$ be the smallest isomorphism class such that there are more components that give the orientation $a \rightarrow b$ to the parent than $b \rightarrow a$ (or vice versa). Then, we define $a \rightarrow b$ to be the reference orientation ($b \rightarrow a$ otherwise). For each isomorphism class $I_j$, we compute now the orientation counters $O_j = (c_j^+, c_j^-)$ such that $c_j^+$ is the number of children in $I_j$ which give the reference orientation and $c_j^-$ is
Recall the example of Figure 4. The graphs $G$ and $H$ have the same triconnected component trees but are not isomorphic. In $S_{(a,b)}$, the 3-bonds form one isomorphism class $I_1$ and the other two components form the second isomorphism class $I_2$, as they all are pairwise isomorphic. The non-isomorphism is detected by comparing the directions given to the parent separating pair. We have $p = 2$ isomorphism classes and for the orientation counters we have $O_1 = O_1' = (0,0)$, whereas $O_2 = (2,0)$ and $O_2' = (1,1)$ and hence $O_2'$ is lexicographically smaller than $O_2$. Therefore we have $T_{(a',b')} <_{\tau} S_{(a,b)}$.

**Complexity.** We argue now, that we can do the four comparison steps in log-space. The first and the second step are similar to Lindell’s algorithm. We define the size of a separating pair node as 2 and the size of a triconnected component as the number of vertices in the component. For the third and fourth step, we have the following cases:

- When we compare two triconnected components $G_i$ and $H_j$, then we have up to four canons. Suppose, we construct and compare two canons $C_g$ and $C_h$ and reach separating pairs $(a,b)$ and $(a',b')$. We store the canons which are not eliminated, which of them $C_g$ and $C_h$ are and the direction of the virtual edges $(a,b)$ and $(a',b')$. Hence, we need $O(1)$ bits.

- When we compare two separating pairs $(a,b)$ and $(a',b')$, then we make a cross-comparison as in Lindell’s algorithm. Hence, we need counters $c_c, c_s, c_\tau$ to store the order profile. This way, we get the isomorphism classes. We further store the orientation counters $O_j$ and $O_j'$ for $I_j$ and $I'_j$. We need $O(\log |I_j|)$ bits on the work-tape for all the counters.

However, we cannot guarantee yet, that the algorithm works in log-space. Let $S_C$ be the subtree rooted at node $C$ in a triconnected component tree. The problem is, that the subtrees (i.e. the children of $C$) where we go into recursion might be of size $> |S_C|/2$, we call this a large child.

To get around this problem, we first check whether the nodes $C$ and $C'$ have a large child. If so then we compare them a priori and store the result of their comparison and the orientation given to the parent. Because $C$ and $C'$ have at most one large child, this needs only $O(1)$ additional bits. In the comparison of $C$ and $C'$, whenever we would go into recursion at those large children, we just look at the work-tape for the result instead.

As seen above, while comparing two trees of size $N$, the algorithm uses no space for making a recursive call for a subtree of size larger than $N/2$, and it
uses $O(\log k_j)$ space if the subtrees are of size at most $N/k_j$, where $k_j \geq 2$. Hence we get the same recurrence for the space $S(N)$ as Lindell:
\[
S(N) \leq \max_j S\left(\frac{N}{k_j}\right) + O(\log k_j),
\]
where $k_j \geq 2$ for all $j$. Thus $S(N) = O(\log N)$. Note that the number $n$ of nodes of $G$ is in general smaller than $N$, because the separating pair nodes occur in all components split off by this pair. But we certainly have $n < N \leq O(n^2)$ [9]. This leads to the following theorem.

**Theorem 4.1.** [6] The isomorphism order between two triconnected component trees of biconnected planar graphs can be computed in log-space.

**The canon.** Once we know the ordering among the subtrees, it is straightforward to output the canon of the triconnected component tree $T$. We traverse $T$ in the tree isomorphism order as in Lindell’s algorithm, outputting the canon of each of the nodes along with virtual edges and delimiters. That is, we output a ‘[’ while going down a subtree, and ‘]’ while going up a subtree.

We need to choose a separating pair as root for the tree. Since there is no distinguished separating pair, we simply cycle through all of them and select the one, which leads to the minimum canon. Let $(a, b)$ be this separating pair. The canonization procedure has two steps. In the first step we compute what we call a canonical list for $S_{(a, b)}$. This is a list of the edges of $G$, also including virtual edges. In the second step we compute the final canon from the canonical list.

**Canon of separating pair nodes.** Consider a subtree $S_{(a, b)}$ rooted at $(a, b)$. We start with computing the reference orientation of $(a, b)$ with oracle calls to the canonical ordering algorithm and output the edge in this direction. Then we recursively output the canonical lists of the subtrees of node $(a, b)$ according to the increasing isomorphism order. Among isomorphic siblings, those which give the reference orientation to the parent come first. We denote this canonical list of edges $l(S, a, b)$. If there is no reference orientation for a child, take the orientation of the parent $(a, b)$.

**Canon of triconnected component nodes.** Consider the subtree $S_{G_i}$ rooted at $G_i$. Let $(a, b)$ be the parent separating pair of $S_{G_i}$ with reference orientation $(a, b)$. If $G_i$ is a 3-bond then output $l(G_i, a, b) = (a, b)$. If $G_i$ is a cycle then output $l(G_i, a, b) = (a, b)(b, v_1)(v_1, v_2)\ldots(v_n, a)$. If $G_i$ is a 3-connected component then compute the minimum of two canons with an oracle call. That is with respect to the given reference orientation $(a, b)$ and both embeddings for $G_i$. Output this canon as $l(G_i, a, b)$. Virtual edges are output in the direction of the reference orientation given to them, if any. Finally, we output the subtrees in the order we have virtual edges in the canon.
4.2 Canonization of planar graphs

Consider the decomposition of a connected planar graph. For each articulation point and biconnected component we define nodes i.e. articulation point nodes and biconnected component nodes. An articulation point node for \( a \) is connected by an edge to the nodes of biconnected components where \( a \) is contained as a vertex. The resulting graph is a tree, the biconnected component tree.

The size of an articulation point node \( a \) is the size of its triconnected component tree.

We give an example. Consider the canonical list \( l(S, a, b) \) of edges for the tree \( S(a, b) \) of Figure 3. Let \( s_i \) be the edge connecting the vertices \( a_i \) with \( b_i \). We also write for short \( l'(S_i, s_i) \) which is one of \( l(S_i, a_i, b_i) \) or \( l(S_i, b_i, a_i) \). The direction of \( s_i \) is as described above. Let \( b_0 = 0 \). Then we have:

\[
l(S, a, b) = \begin{cases} \{ (a, b) \} \cup \{ l(S_{G_1}, a, b) \} \cup \cdots \cup \{ l(S_{G_l}, a, b) \}, & \text{where} \\
l(S_{G_i}, a, b) = \{ l(G_i, a, b) \} \cup \{ l(S_{l_i+1}, a_i, b_i) \} \cup \cdots \cup \{ l'(S_i, s_i) \} \end{cases}
\]

We define the isomorphism order for two biconnected component trees \( S_a \) and \( T_{a'} \) rooted at nodes \( s \) and \( t \) corresponding to articulation points \( a \) and \( a' \) respectively. Also see Figure 5. Let \( |S_a| \) be the sum of the sizes of the nodes in the tree. The size of an articulation point node \( a \) is defined as 1 and the size of a biconnected component node \( B \) is the size of its triconnected component tree \( |T(B)| \).

Let \( S_a \) and \( T_{a'} \) be biconnected component trees rooted at articulation points \( a \) and \( a' \). Define \( S_a \preceq_b T_{a'} \) if

1. \( |S_a| < |T_{a'}| \) or
2. \( |S_a| = |T_{a'}| \) but \( \#s < \#t \) or
3. \( |S_a| = |T_{a'}| \), \( \#s = \#t = k \), but \( (S_{B_1}, \ldots, S_{B_k}) \preceq_b (T_{B_1'}, \ldots, T_{B_k'}) \) lexicographically, where we assume that \( S_{B_1} \preceq_b \cdots \preceq_b S_{B_k} \) and \( T_{B_1'} \preceq_b \cdots \preceq_b T_{B_k'} \) are the ordered subtrees of \( S_a \) and \( T_{a'} \) respectively. To compare the order between the subtrees \( S_{B_i} \) and \( T_{B_i'} \) we compare the triconnected component trees \( T(B_i) \) of \( B_i \) and \( T(B_i') \) of \( B_i' \) and when we reach the first occurrences of some articulation points then we compare recursively the corresponding subtrees rooted at the children of \( B_i \) and \( B_i' \). Note, that these children are again articulation point nodes.

We say that two biconnected component trees are equal, denoted by \( S_a \equiv_b T_{a'} \), if neither of \( S_a \preceq_b T_{a'} \) and \( T_{a'} \preceq_b S_a \) holds. The inductive ordering of the subtrees of \( S_a \) and \( T_{a'} \) proceeds exactly as in Lindell’s algorithm, by partitioning them into size-classes and comparing the children in the same size-class recursively.
We summarize now, how we can compute the isomorphism order when we compare subtrees rooted at articulation points, e.g. \( S_a \) and \( T_{a'} \), and when we compare subtrees rooted at biconnected components, e.g. \( S_{B_i} \) and \( T_{B_j'} \).

Comparing \( S_a \) and \( T_{a'} \) is similar to the case when we compare subtrees rooted at separating pairs in triconnected component trees. We make a cross-comparison of the children and store the counters \( c_\prec, c_\prec=, c_\succ \) for their order profile.

When we compare biconnected components \( B_i \) and \( B_j' \), then we cannot start comparing their biconnected canons. We even cannot compute their canons because we do not have a unique root separating pair for the trees \( T(B_i) \) and \( T(B_j') \). The problem occurs when we have only one fixed vertex in \( B_i \), i.e. the parent articulation point. Datta et. al. bound the number of candidates of root separating pairs of \( T(B_i) \) and \( T(B_j') \). For the detailed case analysis we refer to the paper. Basically, except of some special cases they show that the number of edges is bounded by \( k \), when all the isomorphism classes of the children of \( B_i \) and \( B_j' \) (i.e. children in the biconnected component tree of nodes for \( B_i \) and \( B_j' \)) are of cardinality \( \geq k \).

Hence, all the isomorphism classes contain children \( C \) such that \( |S_C| \leq |S_{B_i}|/k \).

If there is one size class of cardinality one, then we treat this child separately. If there are two or more such size classes, then we even get \( O(1) \) candidates for the root. We will need this in the complexity analysis.

**Complexity according to the biconnected component tree.** First, when we compare articulation points \( a \) and \( a' \) in the biconnected component tree, we have a similar complexity analysis as in Lindell’s algorithm. For the children of \( a \) and \( a' \), we store \( O(\log k) \) bits for isomorphism classes of cardinality \( k \geq 2 \).

Second, when we compare biconnected components \( B \) and \( B' \) in the biconnected component tree then a typical query is of the form \((s, r)\), where \( s \) is the
chosen root of $T(B)$ and $r$ is the index of the edge in the canon, which is to be retrieved. If there are $k$ choices for $T(B)$ and $T(B')$, the base machine cycles through all of them one by one, keeping track of the minimum canon. This takes $O(\log k)$ space. In both cases, we also consider large children (i.e. children $C$ of $B$ such that $|S_C| > |S_B|/2$) a priori. We summarize. If we consider recursively how many bits we store for the roots of biconnected components then we get the recursion equation for the size function.

$$S(N) = \max_j \left\{ S\left(\frac{N}{k_j}\right) + O(\log k_j) \right\}$$

where $k_j \geq 2$. Hence, $S(N) = O(\log N)$.

**Complexity according to the triconnected component trees.** We consider now the comparison of triconnected component trees $T(B)$ and $T(B')$ of biconnected components $B$ and $B'$. In the comparison of $T(B)$ and $T(B')$, we still go into recursion at separating pairs and when we reach virtual edges in canons for triconnected components. What is new, we go into recursion when we reach articulation points. For an example, see Figure 6.

![Figure 6: A biconnected component tree $S_B$ rooted at biconnected component $B$ which has an articulation point $a$ as child, which occurs in the triconnected component tree $T(B)$ of $B$. In $A$ and the other triconnected components the dashed edges are separating pairs.](image-url)

If an articulation point $a$ belongs to many separating pairs, then it can occur in many component nodes in $T(B)$. Recall, that we have a root for the tree. So, there exists a unique component $A$ that is closest to the root, where $a$ is contained. Observe, that the set of component nodes where $a$ is contained is always a connected subtree in $T(B)$. The authors show, that this unique component can be computed in log-space and that the first position where $a$ occurs in the canon of $A$ can be
found in log-space. Exactly there, we go for a into recursion. For all the other occurrences of a we do not go into recursion. Call this the reference copy of a in T(B).

Assume we store separately the bits that we need inside T(B) for all biconnected components B. For this part also a log-space bound can be proved. The size function can therefore be refined. Let C be a node in T(B). The size of the subtree S_C rooted at some node C is the sum of the size of the triconnected subtree rooted at C in T(B), say |S_C| plus the size of all the biconnected subtrees |S_a|, if a is a reference copy of an articulation points in S_C. There is one more special case. If S_a is a large child for B in the biconnected component tree and for C in S_C, then we still go only once into recursion for S_a a priori and store the result. In this case, if a has a reference copy in the subtree of T(B) rooted at C then S_a is not included in the size of S_C. Hence, we get the same recursion equation as before. This finishes the complexity analysis and leads to the following theorem.

**Theorem 4.2.** [6] The isomorphism order between two planar graphs can be computed in log-space.

**The canon.** The canonization of planar graphs proceeds exactly as in the case of biconnected planar graphs. A log-space procedure traverses the biconnected component tree, makes oracle queries to the isomorphism order algorithm and outputs a canonical list of edges, along with delimiters to separate the lists for siblings. A log-space transducer then renames the vertices according to their first occurrence in this list, to get the final canon for the biconnected component tree. This canon depends upon the choice of the root of the biconnected component tree. Further log-space transducers cycle through all the articulation points as roots to find the minimum canon among them, then rename the vertices according to their first occurrence in the canon and finally, remove the virtual edges and delimiters to obtain a canon for the planar graph. This proves the main theorem.

**Theorem 4.3.** [6] A planar graph can be canonized in log-space.

**References**


THE DISTRIBUTED COMPUTING COLUMN

BY

PANAGIOTA FATOUROU

Department of Computer Science, University of Crete
P.O. Box 2208 GR-714 09 Heraklion, Crete, Greece
and
Institute for Computer Science (ICS)
Foundation for Research and Technology (FORTH)
N. Plastira 100. Vassilika Vouton
GR-700 13 Heraklion, Crete, Greece
fatorou@csd.uoc.gr

Advances in Distributed Computing have been simply astonishing during the past few decades. The Distributed Computing Column of BEATCS aims in exposing the community to the most challenging and inspirational results of the field.

The column has been edited for the last eight years by Marios Mavronicolas who did an excellent job on establishing it as a popular venue. Thanks to him but also, and more, to the invited authors who submitted remarkable contributions, the column was operating all these years with big success.

My major objective is to continue projecting the most exciting developments in all areas of Distributed Computing onto the column. Of particular interest are hot, currently emerging topics and technologies, including multi-core algorithms and architectures, transactional memory and other synchronization techniques for ubiquitous parallel programming, self-adaptive, self-organizing and autonomic systems, sensor, mobile, mesh, ad-hoc and peer-to-peer networks and protocols, cluster and grid computing, social networks and game-theoretic approaches, security, fault-tolerance, reliability, and many other stimulating subjects. Emphasis will also be given to the foundations of distributed computing.

This issue, the first that I am editing, and the other issues of this year are devoted to Transactional Memory (TM), the most promising simplification for expressing parallelism which is currently the major obstacle in taking full advantage of the presently dominated multicore chip design. The first article refers
to the theoretical foundations of transactional memory. TM findings under other motivating perspectives will be published in the next issues.

Concluding, let me welcome you to this Column and encourage your submission of articles and other material. I especially welcome surveys providing new insight on fundamental topics of distributed computing, articles describing open problems in challenging areas of the field, and any other material that can bring new, inspirational problems and techniques under the attention of the community. I would be interested to also host reviews of the main conferences or of other interesting events in the field. Suggestions can be sent to me by e-mail.

THE THEORY OF TRANSACTIONAL MEMORY

Rachid Guerraoui
EPFL, Switzerland
rachid.guerraoui@epfl.ch

Michał Kapałka
EPFL, Switzerland
michal.kapalka@epfl.ch

Abstract

Transactional memory (TM) is a promising paradigm for concurrent programming. This paper is an overview of our recent work on defining a theory of TM. We first present a correctness condition of a TM, ensured by most existing TM implementations. Then, we describe two progress properties that characterize the two main classes of TM implementations: obstruction-free and lock-based TMs. We use these properties to establish several results on the inherent power and limitations of TMs.

1 Introduction

Multi-core processors are predicted to be common in home computers, laptops, and maybe even smoke detectors. To exploit the power of modern hardware, applications will need to become increasingly parallel. However, writing scalable concurrent programs is hard and error-prone with traditional locking techniques. On the one hand, coarse-grained locking throttles parallelism and causes lock contention. On the other hand, fine-grained locking is usually an engineering challenge, and as such is not suitable for use by the masses of programmers.

Transactional memory (TM) [28] is a promising technique to facilitate concurrent programming while delivering performance comparable to that of fine-grained locking implementations. In short, a TM allows concurrent threads of
an application to communicate by executing lightweight, in-memory transactions [15]. A transaction accesses shared data and then either commits or aborts. If it commits, its operations are applied to the shared state atomically. If it aborts, however, its changes to the shared data are lost and never visible to other transactions.

The TM paradigm has raised a lot of hope for mastering the complexity of concurrent programming. The aim is to provide the programmer with an abstraction, i.e., the transaction, that makes concurrency as easy as with coarse-grained critical sections, while exploiting the underlying multi-core architectures as efficiently as hand-crafted fine-grained locking. It is thus not surprising to see a large body of work directed at experimenting with various kinds of TM implementation strategies, e.g. [28, 43, 26, 33, 8, 34, 23, 45, 27, 40, 23, 2, 44, 12, 14]. What might be surprising is the little work devoted so far to the formalization of the precise guarantees that TM implementations should provide. Without such formalization, it is impossible to verify the correctness of these implementations, establish any optimality result, or determine whether various TM design trade-offs are indeed fundamental or simply artifacts of certain environments.

From a user’s perspective, a TM should provide the same semantics as critical sections: transactions should appear as if they were executed sequentially, i.e., as if each transaction acquired some global lock for its entire duration. (Remember that the TM goal is to provide a simple abstraction to average programmers.) However, a TM implementation would be inefficient if it never allowed different transactions to run concurrently. Hence, we want to reason formally about executions with interleaving steps of arbitrary concurrent transactions. First, we need a way to state precisely whether a given execution in which a number of transactions execute steps in parallel “looks like” an execution in which these transactions proceed one after the other. That is, we need a correctness condition for TMs. Second, we should define when a TM implementation is allowed to abort a transaction that contends for shared data with concurrent transactions. Indeed, while the ability to abort transactions is essential for all optimistic schemes used by TMs, a TM that abuses this ability by aborting every transaction is, clearly, useless. Hence, we need to define progress properties of TM implementations.

We overview here our work on establishing the theoretical foundations of TMs [20, 19, 21]. We first present opacity—a correctness condition for TMs, which is indeed ensured by most TM implementations, e.g., DSTM [26], ASTM [33], SXM [24], JVSTM [8], TL2 [9], LSA-STM [40], RSTM [34], TinySTM [12], BartokSTM [23], McRT-STM [2], AVSTM [18], and the STM in [39]. The technical challenge in specifying opacity is the ability to reason about concurrent transactional executions in a general and high-level model (a) with arbitrary objects, beyond simple read/write variables, (b) supporting multiple versions of each object (i.e., multi-version protocols, used, e.g., in JVSTM), and
(c) not precluding various TM strategies and optimization techniques, such as invisible reads, lazy updates, or caching.

At first glance, it seems very likely that such a criterion would correspond to one of the numerous ones defined in the literature, e.g., linearizability [29], serializability [37, 5], global atomicity [48], recoverability [22], or rigorous scheduling [7]. However, none of these criteria, nor any straightforward combination or extension thereof, is sufficient to describe the semantics of TM with its subtleties [20]. In particular, none of them captures exactly the very requirement that every transaction, including a live (i.e., not yet completed) one, accesses a consistent state, i.e., a state produced by a sequence of previously committed transactions. A live transaction that accesses an inconsistent state might create significant dangers when executed within a general TM framework [45]. It is thus not surprising that most TM implementations employ mechanisms that disallow such situations, sometimes at a big cost. At a very high abstraction level, disallowing transactions to access inconsistent states resembles, in the database terminology, preventing dirty reads or, more generally, the read skew phenomenon [4], when generalized to all transactions (not only committed ones as in [4]) and arbitrary objects. Capturing this intuitive idea in a precise manner is not trivial.

It is worth noting that the difference between opacity and classical database properties like serializability is not “cosmetic”. In fact, we showed in [20] that opacity is inherently more expensive to implement than the combination of (strict) serializability [37, 5] (or global atomicity [48]) and the strongest form of recoverability [22]. Basically, we proved a complexity lower bound that holds for TMs that ensure opacity, but does not hold for TMs that ensure serializability and recoverability.

We also present in this paper progress properties of the two main classes of existing TM implementations: obstruction-free [26] and lock-based ones. The intuition behind the progress semantics of such TMs has been known, but precise definitions were missing.

Roughly speaking, an obstruction-free TM (OFTM), such as DSTM, ASTM, RSTM, or SXM, guarantees progress for every thread of an application that executes transactions alone (i.e., without contention) for sufficiently long time. OFTMs are appealing in real-time systems, where priority inversion is an important issue, or within operating systems, where kernel-level transactions (e.g., inside interrupt handlers) must be able to preempt (and, in many cases, abort) user-level ones at any time [46]. In an OFTM, a transaction that is preempted, delayed, or even crashed cannot inhibit the progress of other transactions. This means that an OFTM cannot internally use any blocking mechanisms such as critical sections.

Many TM implementations that are considered effective, e.g., TL2, TinySTM, a version of RSTM, BartokSTM, or McRT-STM are, however, not obstruction-
free. They internally use locking, in order to reduce the overheads of TM mechanisms. We define the progress semantics of lock-based TMs by introducing a property, which we call strong progressiveness,\(^1\) and which stipulates the two following requirements: (1) A transaction that encounters no conflict must be able to commit (basically, a conflict occurs when two or more concurrent transactions access the same transactional variable and at least one of those accesses is not read-only); (2) If a number of transactions have only a “simple” conflict, i.e., on a single transactional variable, then at least one of them must be able to commit. Requirement (1) captures the common intuition about the progress of any TM (see [41]). Requirement (2) ensures that conflicts that are easy to resolve do not cause all conflicting transactions to be aborted. This is especially important when non-transactional accesses to shared variables are encapsulated inside unit transactions to ensure strong atomicity [6]. Strong progressiveness is ensured by state of the art lock-based implementations, such as TL2, TinySTM, a version of RSTM, BartokSTM, and McRT-STM.\(^2\)

We also give in this paper an overview of the theoretical results proved for OFTMs and strongly progressive (lock-based) TMs that ensure opacity.

It is important to notice that the paper is only an overview of previously published results. In particular, we do not give here any proofs of the theorems and lemmas that we state. Those proofs, as well as further details and discussions of the results presented here, can be found in [20, 19, 21].

2 Preliminaries

2.1 Shared Objects and their Implementations

Processes and objects. We consider a classical asynchronous shared-memory system [25, 32] of \(n\) processes \(p_1, \ldots, p_n\) that communicate by executing operations on (shared) objects. An example of a very simple object is a register, which exports only read and write operations. Operation read returns the current state (value) of the register, and operation write(v) sets the state of the register to value v.

An execution of every operation is delimited by two events: the invocation of the operation and the response from the operation. We assume that, in every run of the system, all events can be totally ordered according to their execution time.

---

\(^1\)We call it “strong” by opposition to a weaker form of progressiveness that we also introduce in [21].

\(^2\)The source code of the implementations of BartokSTM and McRT-STM is not publicly available. We could thus verify strong progressiveness of those TMs only from their algorithm descriptions in [23] and [2], respectively.
Process $p_i$

Events

Steps

Figure 1: An example execution of an operation $trans$ on a shared object $A$ by a process $p_i$. Operation $trans$ is implemented using operations $inc$ and $dec$ on base objects $x$ and $y$.

If several events are executed at the same time (e.g., on multiprocessor systems), then they can be ordered arbitrarily. We call a pair of invocation of an operation and the subsequent response from this operation an operation execution.

A object $x$ may be provided either directly in hardware, or implemented from other, possibly more primitive, base objects (cf. Figure 1). We call the events of operations on base objects steps. We assume that each process executes operations on shared objects, and on base objects, sequentially.

**Wait-freedom.** We focus on object implementations that are wait-free [25]. Intuitively, an implementation of an object $x$ is wait-free if a process that invokes an operation on $x$ is never blocked indefinitely long inside the operation, e.g., waiting for other processes. Hence, processes can make progress independently of each other. More precisely, an implementation $I_x$ of an object $x$ is wait-free, if whenever any process $p_i$ invokes an operation on $I_x$, $p_i$ returns from the operation within a finite number of its own steps.

**Computational equivalence.** We say that object $x$ can implement object $y$ if there exists an algorithm that implements $y$ using some number of instances of $x$ (i.e., a number of base objects of the same type as $x$) and atomic (i.e., linearizable [29]) registers. We say that objects $x$ and $y$ are equivalent if $x$ can implement $y$ and $y$ can implement $x$.

**The power of a shared object.** We use the consensus number [25] as a metric of the power of objects. The consensus number of an object $x$ is the maximum number of processes among which one can solve (wait-free) consensus using any number of instances of $x$ (i.e., base objects of the same type as $x$) and atomic registers.

The consensus problem consists for a number of processes to agree (decide) on a single value chosen from the set of values these processes have proposed. It is known that, in an asynchronous system, implementing wait-free consensus is impossible when only registers are available [13]. Solving consensus consists in ensuring the following properties: (1) every value decided is one of the values proposed (validity); and (2) no two processes decide different values (agreement).
2.2 Transactional Memory (TM)

A TM enables processes to communicate by executing transactions. A transaction may perform operations on objects shared with other transactions, called transactional objects (or t-objects, for short), as well as local computations on objects inaccessible to other transactions. For simplicity, we will say that a transaction $T$ performs some action, meaning that the process executing $T$ performs this action within the transactional context of $T$. We will call t-variables those t-objects that are registers, i.e., that provide only read and write operations.

Every transaction has a unique identifier (e.g., $T_1$, $T_2$, etc.). (We use the terms “transaction” and “transaction identifier” interchangeably.) Every transaction, upon its first action, is initially live and may eventually become either committed or aborted, as explained in the following paragraphs. A transaction that is not live does no longer perform any actions. Retrying an aborted transaction (i.e., the computation the transaction intends to perform) is considered in our model as a new transaction, with a different transaction identifier.

**TM as a shared object.** A TM can be viewed as an object with operations that allow for the following: (1) Executing any operation on a t-object $x$ within a transaction $T_k$ (returns the response of the operation or a special value $A_k$); (2) Requesting transaction $T_k$ to be committed (operation $tryC(T_k)$ that returns either $A_k$ or $C_k$); (3) Requesting transaction $T_k$ to be aborted (operation $tryA(T_k)$ that always returns $A_k$). The special return value $A_k$ (abort event) is returned by a TM to indicate that transaction $T_k$ has been aborted. The return value $C_k$ (commit event) is a confirmation that $T_k$ has been committed.

As for other objects, we assume that every implementation of a TM is wait-free, i.e., that the individual operations of transactions are wait-free. This is indeed the case for most TM implementations (including lock-based ones; see [21]).

If $x$ is a t-object (provided by a given TM), then we denote by $inv(x,op)$, $ret_k(x,op \rightarrow v)$, and $x,op_k \rightarrow v$ an invocation, response, and execution (invocation and the subsequent response), respectively, of operation $op$ on $x$ by transaction $T_k$ (returning value $v$). We also denote by $A_k$ (and $C_k$) an abort (commit) event of transaction $T_k$.

**Histories.** Consider any TM and any run. A history (of the TM) is a sequence of invocation and response events of operations executed by processes on the TM in this run. Let $M$ be any implementation of the TM. An implementation history of $M$ is the sequence of (1) invocation and response events of operations executed by processes on $M$, and (2) the corresponding steps of $M$ in a given run.

Let $H$ be any (implementation) history of a TM and $T_k$ be any transaction. We denote by $H|T_k$ the sequence of all events executed by $T_k$ in $H$. We say that $T_k$ is in $H$, and write $T_k \in H$, if there is some event executed by $T_k$ in $H$, i.e., if $H|T_k$ is a non-empty sequence. We also denote by $H|x$ the restriction of $H$ to events
executed on t-object $x$.

We say that transaction $T_k$ is committed (respectively, aborted) in $H$, if $H$ contains commit event $C_k$ (resp., abort event $A_k$). A transaction that is neither committed nor aborted is called live. We say that transaction $T_k$ is forceably aborted in $H$, if $T_k$ is aborted in $H$ but there is no invocation of operation $\text{tryA}(T_k)$ in $H$. We say that $T_k$ is commit-pending in $H$, if $H$ contains an invocation of operation $\text{tryC}(T_k)$ but $T_k$ is still live in $H$.

We say that a transaction $T_k$ precedes a transaction $T_m$ in history $H$, and write $T_k \prec_H T_m$, if $T_k$ is completed and the last event of $T_k$ precedes (in $H$) the first event of $T_m$. We say that transactions $T_k$ and $T_m$ are concurrent in a history $H$, if neither $T_k$ precedes $T_m$, nor $T_m$ precedes $T_k$ (in $H$).

We say that history $H$ is sequential if no two transactions in $H$ are concurrent. We say that $H$ is complete if $H$ does not contain any live transaction.

We assume that every transaction $T_k$ in $H$ is executed by a single process. Conversely, we assume that every process $p_i$ executes only one transaction at a time, i.e., that no two transactions are concurrent at any given process. Note also that, because a completed transaction does not perform any further action, a commit/abort event (if any) of every transaction $T_k$ in $H$ must be the last event in sub-history $H|T_k$.

**Sequential specification of a t-object.** We use the concept of a *sequential specification* to describe the semantics of t-objects, as in [48, 29]. Intuitively, a sequential specification of a t-object $x$ lists all sequences of operation executions on $x$ that are considered correct when executed outside any transactional context, e.g., in a standard, single-threaded application. For example, the sequential specification of a t-variable $x$, denoted by $\text{Seq}(x)$, is the set of all sequences of read and write operation executions on $x$, such that in each sequence that belongs to $\text{Seq}(x)$, every read (operation execution) returns the value given as an argument to the latest preceding write (regardless transaction identifiers). (In fact, $\text{Seq}(x)$ also contains sequences that end with a pending invocation of read or write, but this is a minor detail.) Such a set defines precisely the semantics of a t-variable in a single-threaded, non-transactional system.

More formally, let an *object-local history* of a t-object $x$ be any prefix $S$ of a sequence of operation executions, such that $S|x = S$. Then, a sequential specification $\text{Seq}(x)$ of a t-object $x$ can be any prefix-closed set of object-local histories of $x$. (A set $Q$ of sequences is *prefix-closed* if, whenever a sequence $S$ is in $Q$, every prefix of $S$ is also in $Q$.)

---

3An operation execution specifies a transaction identifier, but the identifier can be treated as part of the arguments of the executed operation. In fact, in most cases, the semantics of an operation does not depend on the transaction that issues this operation.
Opacity is a safety property that captures the intuitive requirements that (1) all operations performed by every committed transaction appear as if they happened at some single, indivisible point during the transaction lifetime, (2) no operation performed by any aborted transaction is ever visible to other transactions (including live ones), and (3) every transaction always observes a consistent state of the system.

We start by explaining informally, step by step, how one can determine whether a given history of a TM ensures opacity. Next, we give a precise definition of opacity and the related terms, and provide an example.

3.1 Opacity Step by Step

To help understand the definition of opacity, we first consider very simple histories, and increase their complexity step by step. The precise definitions of the terms that correspond to the steps described here are given in Section 3.2.

Opacity is trivial to express and verify for sequential histories in which every transaction, except possibly the last one, is committed. Basically, if \( S \) is such a history, then \( S \) is considered correct, and called legal, if, for every t-object \( x \), the sub-history \( S|x \) respects the semantics of \( x \), i.e., \( S|x \) belongs to the sequential specification of \( x \). For example, if a transaction \( T_i \) writes value \( v \) to a t-variable \( x \) at some point in history \( S \), then all subsequent reads of \( x \) in \( S \), performed by \( T_i \) or by a following transaction, until the next write of \( x \), must return value \( v \).

The situation becomes more difficult if \( S \) is sequential but contains some aborted transactions followed by committed ones. For example, if an aborted transaction \( T_i \) writes value \( v \) to a t-variable \( x \) (and no other transaction writes \( v \) to \( x \)), then only \( T_i \) can read \( v \) from \( x \) thereafter. A read operation on \( x \) executed by a transaction following \( T_i \) must return the last value written to \( x \) by a preceding committed transaction. Basically, when considering a transaction \( T_i \) (committed or aborted) in \( S \), we have to remove all aborted transactions that precede \( T_i \) in \( S \). We then say that \( T_i \) is legal in \( S \), if \( T_i \) together with all committed transactions preceding \( T_i \) in \( S \) form a legal history. Clearly, for an arbitrary sequential history \( S \) to be correct, all transactions in \( S \) must be legal.

To determine the opacity of an arbitrary history \( H \), we ask whether \( H \) “looks like” some sequential history \( S \) that is correct (i.e., in which every transaction is legal). In the end, a user of a TM should not observe, or deal with, concurrency between transactions. More precisely, history \( S \) should contain the same transactions, performing the same operations, and receiving the same return values from those operations, as history \( H \). We say then that \( H \) is equivalent to \( S \). Equivalent histories differ only in the relative position of events of different transactions.
Moreover, the real-time order of transactions in $H$ should be preserved in $S$.

There is, however, one problem with finding a sequential history that is equivalent to a given history $H$: if two or more transactions are live in $H$, then there is no sequential history that is equivalent to $H$. Basically, if $S$ is a sequential history, then $≺_S$ must be a total order; however, if a transaction $T_i$ precedes a transaction $T_k$ in $S$, i.e., if $T_i ≺_S T_k$, then $T_i$ must be committed or aborted. To solve the problem, observe that the changes performed by a transaction $T_i$ should not become visible to other transactions until $T_i$ commits. Transaction $T_i$ commits at some point (not visible to the user) between the invocation and the response of operation $\text{tryC}(T_i) \rightarrow C_i$. That is, the semantics of $T_i$ is the same as of an aborted transaction until $T_i$ invokes $\text{tryC}(T_i)$, but this semantics might change (to the one of a committed transaction) at any point in time after $T_i$ becomes commit-pending. Hence, we can safely transform an arbitrary history $H$ into a complete history $H'$ by (1) aborting all live and non-commit-pending transactions in $H$, and (2) committing or aborting every commit-pending transaction in $H$.

### 3.2 Definition of Opacity

Let $S$ be any sequential history such that every transaction in $S$, except possibly the last one, is committed. We say that $S$ is legal if, for every $t$-object $x$, the subsequence $S|_x$ is in set $\text{seq}(x)$ (the sequential specification of $x$).

Let $S$ be any complete sequential history. We denote by $\text{visible}_S(T_i)$ the longest subsequence $S'$ of $S$ such that, for every transaction $T_k \in S'$, either (1) $k = i$, or (2) $T_k$ is committed and $T_k ≺_S T_i$. We say that a transaction $T_i \in S$ is legal in $S$, if history $\text{visible}_S(T_i)$ is legal.

Let $H$ and $H'$ be any histories. We say that $H$ and $H'$ are equivalent if, for every transaction $T_i$, $H|T_i = H'|T_i$. We say that history $H'$ preserves the real-time order of (equivalent) history $H$, if $≺_H \subseteq ≺_{H'}$. That is, if $T_i ≺_H T_j$, then $T_i ≺_{H'} T_j$, for any two transactions $T_i$ and $T_j$ in $H$.

Intuitively, a completion of a history $H$ is any history $H'$ obtained from $H$ by aborting or committing every commit-pending transaction in $H$, and by aborting every other live transaction in $H$. More precisely, a completion of a history $H$ is any valid history $H'$ of the form $H \cdot Q$, where $Q$ is a sequence of invocations of operation $\text{tryC}$, commit events, and abort events, such that (1) every transaction that is live and not commit-pending in $H$ is aborted in $H'$, and (2) every transaction that is commit-pending in $H$ is either committed or aborted in $H'$. In particular, the only completion of a complete history $H$ is $H$ itself.

**Definition 1.** A history $H$ is opaque if there exists a sequential history $S$ equivalent to any completion of $H$, such that (1) $S$ preserves the real-time order of $H$, and (2) every transaction $T_i \in S$ is legal in $S$. 


Note that the definition of opacity does not require every prefix of an opaque history to be also opaque. Thus, the set of all opaque histories is not prefix-closed. For example, while the following history is opaque:

\[ H = \langle x.\text{write}(1), \ x.\text{read}_2 \rightarrow 1, \ \text{tryC}(T_1) \rightarrow C_1, \ \text{tryC}(T_2) \rightarrow C_2 \rangle, \]

the prefix \( H' = \langle x.\text{write}(1), \ x.\text{read}_2 \rightarrow 1 \rangle \) of \( H \) is not opaque (assuming the initial value of \( x \) is 0), because, in \( H' \), transaction \( T_2 \) reads value written by \( T_1 \) that is not committed or commit-pending. However, a history of a TM is generated progressively and at each time the history of all events issued so far must be opaque. Hence, there is no need to enforce prefix-closeness in the definition of opacity, which should be as simple as possible.

The way we define the real-time ordering between transactions introduces a subtlety to the definition of opacity. Basically, the following situation is possible (and considered correct): a transaction \( T_1 \) updates some t-object \( x \), and then some other transaction \( T_2 \) concurrent to \( T_1 \) observes an old state of \( x \) (from before the update of \( T_1 \)) even after \( T_1 \) commits. For example, consider the following history (\( x \) and \( y \) are t-variables with initial value 0):

\[ H = \langle x.\text{read} \rightarrow 0, \ x.\text{write}(5), \ y.\text{write}(5), \ \text{tryC}(T_1) \rightarrow A_1, \ \text{tryC}(T_2) \rightarrow C_2, \ y.\text{read} \rightarrow 5, \ y.\text{read} \rightarrow 0 \rangle. \]

In \( H \), transaction \( T_1 \) appears to happen before \( T_2 \), because \( T_1 \) reads the initial values of t-variables \( x \) and \( y \) that are modified by \( T_2 \). Transaction \( T_3 \), on the other hand, appears to happen after \( T_2 \), because it reads the value of \( y \) written by \( T_2 \). Consider the following sequential history:

\[ S = \langle x.\text{read} \rightarrow 0, \ y.\text{read} \rightarrow 0, \ \text{tryC}(T_1) \rightarrow A_1, \ x.\text{write}(5), \ y.\text{write}(5), \ \text{tryC}(T_2) \rightarrow C_2, \ y.\text{read} \rightarrow 5, \ \text{tryC}(T_3) \rightarrow A_3 \rangle. \]

It is easy to see that \( S \) is equivalent to the completion \( H \cdot \langle \text{tryC}(T_1) \rightarrow A_1, \ \text{tryC}(T_3) \rightarrow A_3 \rangle \) of \( H \), and that \( S \) preserves the real-time order of \( H \). As, clearly, every transaction is legal in \( S \), history \( H \) is opaque.

However, at first, it may seem wrong that the \textit{read} operation of transaction \( T_3 \) returns the value written to \( y \) by the committed transaction \( T_2 \), while the following \textit{read} operation, by transaction \( T_1 \), returns the old value of \( y \). But if \( T_1 \) read value 5 from \( y \), then opacity would be violated. This is because \( T_1 \) would observe an inconsistent state of the system: \( x = 0 \) and \( y = 5 \). Thus, letting \( T_1 \) read 0 from \( y \) is the only way to prevent \( T_1 \) from being aborted without violating opacity. Multi-version TMs, like JVSTM and LSA-STM, indeed use such optimizations to allow long read-only transactions to commit despite concurrent updates performed by
other transactions. In general, it seems that forcing the order between operation executions of different transactions to be preserved, in addition to the real-time order of transactions themselves, would be too strong a requirement.

3.3 Example of an Opaque History

To illustrate our definition, consider the following history $H$, of three transactions accessing two t-variables ($x$ and $y$), corresponding to the execution depicted in Figure 2:

$$H = \langle x.\text{write}(1), y.\text{write}(2), \text{commit}, x.\text{read} \rightarrow 1, x.\text{write}(5), y.\text{read} \rightarrow 2, \text{abort}, y.\text{write}(3), x.\text{read} \rightarrow 1, \text{commit} \rangle.$$

Clearly, the only completion of $H$ is $H$ itself as there is no live transaction in $H$. Moreover, $\prec_H = \{(T_2, T_3)\}$ because $T_1$ is concurrent with $T_2$ and $T_3$. Therefore, we can find three sequential histories that are equivalent to $H$ and preserve the real-time order of $H$ (relation $\prec_H$). However, $T_1$ reads from $x$ the value that has been written by committed transaction $T_2$. Thus, a sequential history in which $T_1$ precedes $T_2$ is not legal. Similarly, $T_3$ cannot precede $T_1$: $T_1$ reads from $y$ the value written by $T_2$ and not the value written by the committed transaction $T_3$. Consider the following sequential history $S = H[T_2] \cdot H[T_1] \cdot H[T_3]$. Clearly, $S$ is equivalent to $H$ and preserves the real-time order of $H$. Furthermore, every transaction is legal in $S$, because sequential histories $\text{visible}_H(T_2) = H[T_2], \text{visible}_H(T_1) = H[T_2] \cdot H[T_1], \text{and visible}_H(T_3) = H[T_2] \cdot H[T_3]$ are all legal. Therefore, history $H$ is opaque.

4 Obstruction-Free TMs

In this section, we define precisely the class of obstruction-free TMs (OFTMs). We also determine the consensus number of OFTMs and show an inherent limitation of those TMs.
Our definition of an OFTM is based on the formal description of obstruction-free objects from [3]. In [19], we consider alternative definitions but we show, however, that these are computationally equivalent to the one we give here.

4.1 Definition of an OFTM

The definition we consider here uses the notion of step contention [3]: it says, intuitively, that a transaction $T_k$ executed by a process $p_i$ can be forcefully aborted only if some process other than $p_i$ executed a step of the TM implementation concurrently to $T_k$.

More precisely, let $E$ be any implementation history of any TM implementation $M$. We say that a transaction $T_k$ executed by a process $p_i$ encounters step contention in $E$, if there is a step of $M$ executed by a process other than $p_i$ in $E$ after the first event of $T_k$ and before the commit or abort event of $T_k$ (if any).

**Definition 2.** We say that a TM implementation $M$ is obstruction-free (i.e., is an OFTM) if in every implementation history $E$ of $M$, and for every transaction $T_k \in E$, if $T_k$ is forcefully aborted in $E$ then $T_k$ encounters step contention in $E$.

4.2 The Power of an OFTM

We show that the consensus number of an OFTM is 2. We do so by first exhibiting an object, called $fo$-consensus, that is equivalent to any OFTM, and then showing that the consensus number of $fo$-consensus is 2.

Intuitively, $fo$-consensus (introduced in [3] as “fail-only” consensus) provides an implementation of consensus (via an operation $propose$), but allows $propose$ to abort when it cannot return a decision value because of concurrent invocations of $propose$. When $propose$ aborts, it means that the operation did not take place, and so the value proposed using this operation has not been “registered” by the $fo$-consensus object (recall that only a value that has been proposed, and “registered”, can be decided). A process which $propose$ operation has been aborted may retry the operation many times (possibly with different proposed value), until a decision value is returned.

More precisely, let $D$ be any set, such that $\bot \notin D$. $fo$-consensus (object) implements a single operation, called $propose$, that takes a value $v \in D$ as an argument and returns a value $v' \in D \cup \{\bot\}$. If a process $p_i$ is returned a non-$\bot$ value $v'$ from $propose(v)$, we say that $p_i$ decides value $v'$. Once $p_i$ decides some value, $p_i$ does not invoke $propose$ anymore. When operation $propose$ returns $\bot$, we say that the operation aborts.

Consider any implementation $I$ of a $fo$-consensus object. We say that an execution of operation $propose$ of $I$ by a process $p_i$ is step contention-free if there is
no step of $I$ executed by a process other than $p_i$ between the invocation and the
response events of this operation execution. Fo-consensus satisfies the following
properties: (1) if some process decides value $v$, then $v$ is proposed by some propose operation that does not abort; (2) no two processes decide different values; and (3) if a propose operation is step contention-free, then the operation does not abort.

Theorem 3. An OFTM is equivalent to fo-consensus.

Theorem 4. Fo-consensus cannot implement (wait-free) consensus in a system of 3 or more processes.

From Theorem 3, Theorem 4, and the claim of [3] that consensus can be implemented from fo-consensus and registers in a system of 2 processes, we have:

Theorem 5. The consensus number of an OFTM is 2.

Corollary 6. There is no algorithm that implements an OFTM using only registers.

4.3 An Inherent Limitation of OFTMs

We show that no OFTM can be strictly disjoint-access-parallel. To define the
notion of strict disjoint-access-parallelism, we distinguish operations that modify
the state of a base object, and those that are read-only. We say that two processes
(or transactions executed by these processes) conflict on a base object $x$, if both
processes execute each an operation on $x$ and at least one of these operations
modifies the state of $x$.

Intuitively, a TM implementation $M$ is strictly disjoint-access-parallel if it en-
sures that processes executing transactions which access disjoint sets of t-objects
do not conflict on common base objects (used by $M$). More precisely:

Definition 7. We say that a TM implementation $M$ is strictly disjoint-access-
parallel if, for every implementation history $E$ of $M$ and every two transactions $T_i$ and $T_k$ in $E$, if $T_i$ and $T_k$ conflict on some base object, then $T_i$ and $T_k$ both access some common t-object.

Theorem 8. No OFTM is strictly disjoint-access-parallel.

It is worth noting that the original notion of disjoint-access-parallelism, in-
troduced in [30], allows for transactions that are indirectly connected via other transactions to conflict on common base objects. For example, if a transaction $T_1$
accesses a t-object $x$, $T_2$ accesses $y$, and $T_3$ accesses both $x$ and $y$, then there is a
dependency chain from $T_1$ to $T_2$ via $T_3$, even though the two transactions $T_1$ and
The Bulletin of the EATCS

$T_2$ use different t-objects. Disjoint-access-parallelism allows then the processes executing $T_1$ and $T_2$ to conflict on some base objects. Disjoint-access-parallelism in the sense of [30] can be ensured by an OFTM implementation, e.g., DSTM.

It is also straightforward to implement a TM that is strictly disjoint-access-parallel but not obstruction-free, e.g., using two-phase locking [11] or the TL algorithm [10].

5 Lock-Based TMs

Lock-based TMs are TM implementations that use (internally) mutual exclusion to handle some phases of a transaction. Most of them use some variant of the two-phase locking protocol, well-known in the database world [11].

From the user’s perspective, however, the choice of the mechanism used internally by a TM implementation is not very important. What is important is the semantics the TM manifests on its public interface, and the time/space complexities of the implementation. If those properties are known, then the designer of a lock-based TM is free to choose the techniques that are best for a given hardware platform, without the fear of breaking existing applications that use a TM.

In this section, we define strong progressiveness—a progress property commonly ensured by lock-based TMs. We determine the consensus number of strongly progressive TMs, and show an inherent performance trade-off in those TMs.

For simplicity of presentation, we assume in this section that all t-objects are t-variables. That is, they export only read and write operations. We discuss how to deal with arbitrary t-objects in Section 5.4.

5.1 Strong Progressiveness

Intuitively, strong progressiveness says that (1) if a transaction has no conflict then it cannot be forcefully aborted, and (2) if a group of transactions conflict on a single t-variable, then not all of those transactions can be forcefully aborted. Roughly speaking, concurrent transactions conflict if they access the same t-variable in a conflicting way, i.e., if at least one of those accesses is a write operation.\footnote{We assume no false conflicts here; we discuss this assumption in Section 5.4.}

Strong progressiveness is not the strongest possible progress property. The strongest one, which requires that no transaction is ever forcefully aborted, cannot be implemented without throttling significantly the parallelism between transactions, and is thus impractical in multi-processor systems.

Strong progressiveness, however, still gives a programmer the following important advantages. First, it guarantees that if two independent subsystems of an...
application do not share any memory locations (or t-variables), then their trans-
actions are completely isolated from each other (i.e., a transaction executed by
a subsystem $A$ does not cause a transaction in a subsystem $B$ to be forcefully
aborted). Second, it avoids “spurious” aborts: the cases when a transaction can
abort are strictly defined. Third, it ensures global progress for single-operation
transactions, which is important when non-transactional accesses to t-variables
are encapsulated into transactions in order to ensure strong atomicity [6]. Finally,
it ensures that processes are able to eventually communicate via transactions (al-
beit in a simplified manner—through a single t-variable at a time). Nevertheless,
one can imagine many other reasonable progress properties, for which strong pro-
gressiveness can be a good reference point.

More precisely, let $H$ be any history of a TM and $T_i$ be any transaction in $H$. We
denote by $WSet_H(T_k)$ and $RSet_H(T_k)$ the sets of t-variables on which $T_k$ exe-
cuted, respectively, a write or a read operation in $H$. We denote by $RWSet_H(T_k)$
the union of sets $RSet_H(T_k)$ and $WSet_H(T_k)$, i.e., the set of t-variables accessed
(read or written) by $T_k$ in history $H$. We say that two transactions $T_i$ and $T_k$ in $H$
conflict on a t-variable $x$, if (1) $T_i$ and $T_k$ are concurrent in $H$, and (2) either $x$
is in $WSet_H(T_k)$ and in $RWSet_H(T_i)$, or $x$ is in $WSet_H(T_i)$ and in $RWSet_H(T_k)$. We
say that $T_k$ conflicts with a transaction $T_i$ in $H$ if $T_i$ and $T_k$ conflict in $H$ on some
t-variable.

Let $H$ be any history, and $T_i$ be any transaction in $H$. We denote by $CVar_H(T_i)$
the set of t-variables on which $T_i$ conflicts with any other transaction in history $H$. That is, a t-variable $x$ is in $CVar_H(T_i)$ if there exists a transaction $T_k \in H$, $k \neq i$, such that $T_i$ conflicts with $T_k$ on t-variable $x$.

Let $Q$ be any subset of the set of transactions in a history $H$. We denote by $CVar_H(Q)$ the union of sets $CVar_H(T_i)$ for all $T_i \in Q$.

Let $CTrans(H)$ be the set of subsets of transactions in a history $H$, such that a set $Q$ is in $CTrans(H)$ if no transaction in $Q$ conflicts with a transaction not in $Q$. In particular, if $T_i$ is a transaction in a history $H$ and $T_j$ does not conflict with any other transaction in $H$, then $\{T_i\} \in CTrans(H)$.

Definition 9. A TM implementation $M$ is strongly progressive, if in every his-
tory $H$ of $M$ the following property is satisfied: for every set $Q \in CTrans(H)$, if $|CVar_H(Q)| \leq 1$, then some transaction in $Q$ is not forcefully aborted in $H$.

5.2 The Power of a Lock-Based TM

We show here that the consensus number of a strongly progressive TM is 2. First,
we prove that a strongly progressive TM is computationally equivalent to a strong
try-lock object that we define in this section. That is, one can implement a strongly
progressive TM from (a number of) strong try-locks and registers, and vice versa.

Second, we determine that the consensus number of a strong try-lock is 2.

All lock-based TMs we know of use (often implicitly) a special kind of locks, usually called try-locks [42]. Intuitively, a try-lock is an object that provides mutual exclusion (like a lock), but does not block processes indefinitely. That is, if a process \( p_i \) requests a try-lock \( L \), but \( L \) is already acquired by a different process, \( p_i \) is returned the information that its request failed instead of being blocked waiting until \( L \) is released.

Try-locks keep the TM implementation simple and avoid deadlocks. Moreover, if any form of fairness is needed, it is provided at a higher level than at the level of individual locks—then more information about a transaction can be used to resolve conflicts and provide progress. Ensuring safety and progress can be effectively separate tasks.

More precisely, a try-lock is an object with the following operations: (1) trylock, that returns true or false; and (2) unlock, that always returns ok. Let \( L \) be any try-lock. If a process \( p_i \) invokes trylock on \( L \) and is returned true, then we say that \( p_i \) has acquired \( L \). Once \( p_i \) acquires \( L \), we say that (1) \( p_i \) holds \( L \) until \( p_i \) invokes operation unlock on \( L \), and (2) \( L \) is locked until \( p_i \) returns from operation unlock on \( L \). (Hence, \( L \) might be locked even if no process holds \( L \)—when some process that was holding \( L \) is still executing operation unlock on \( L \).)

Every try-lock \( L \) guarantees the following property, called mutual exclusion: no two processes hold \( L \) at the same time.

Intuitively, we say that a try-lock \( L \) is strong if whenever several processes compete for \( L \), then one should be able to acquire \( L \). This property corresponds to deadlock-freedom, livelock-freedom, or progress [38] properties of (blocking) locks.

**Definition 10.** We say that a try-lock \( L \) is strong, if \( L \) ensures the following property, in every run: if \( L \) is not locked at some time \( t \) and some process invokes operation trylock on \( L \) at \( t \), then some process acquires \( L \) after \( t \).

While there exists a large number of lock implementations, only a few are try-locks or can be converted to try-locks in a straightforward way. The technical problems of transforming a queue (blocking) lock into a try-lock are highlighted in [42]. It is trivial to transform a typical TAS or TATAS lock [38] into a strong try-lock [21].

**Theorem 11.** A strongly progressive TM is equivalent to a strong try-lock.

**Theorem 12.** A strong try-lock has consensus number 2.

Hence, by Theorem 11 and Theorem 12, the following theorem holds:
Theorem 13. A strongly progressive TM has consensus number 2.

Corollary 14. There is no algorithm that implements a strongly progressive TM using only registers.

5.3 Performance Trade-Off in Lock-Based TMs

We show that the space complexity of every strongly progressive TM that uses invisible reads is at least exponential with the number of t-variables available to transactions. The invisible reads strategy is used by a majority of lock-based TM implementations [9, 34, 23, 2, 12] as it allows efficient optimistic reading of t-variables. Intuitively, if invisible reads are used, a transaction that reads a t-variable does not write any information to base objects. Hence, many processors can concurrently execute transactions that read the same t-variables, without invalidating each other’s caches and causing high traffic on the inter-processor bus. However, transactions that update t-variables do not know whether there are any concurrent transactions that read those variables. (For a precise definition of invisible reads, consult [21].)

The size of a t-variable or a base object $x$ can be defined as the number of distinct, reachable states of $x$. In particular, if $x$ is a t-variable or a register object, then the size of $x$ is the number of values that can be written to $x$. For example, the size of a 32-bit register is $2^{32}$.

Theorem 15. Every strongly progressive TM implementation that uses invisible reads and provides to transactions $N$ t-variables of size $K_s$ uses $\Omega\left(\frac{N}{K_b}\right)$ base objects of size $K_b$.

This result might seem surprising, since it is not obvious that modern lock-based TMs have non-linear space complexity. The exponential (or, in fact, unbounded) complexity comes from the use of timestamps that determine version numbers of t-variables. TM implementations usually reserve a constant-size word for each version number (which gives linear space complexity). However, an overflow can happen and has to be handled in order to guarantee opacity. This requires (a) limiting the progress (strong progressiveness) of transactions when overflow occurs, and (b) preventing read-only transactions from being completely invisible [21]. Concretely speaking, our result means that efficient TM implementations (the ones that use invisible reads) must either intermittently (albeit very rarely) violate progress guarantees, or use unbounded timestamps.

---

5In fact, the result holds also for TMs that ensure a property called weak progressiveness, which is strictly weaker than strong progressiveness [21].
5.4 Generalizing Strong Progressiveness

The two major assumptions we made when defining the notion of strong progressiveness were that all t-objects are t-variables (i.e., they support only read and write operations), and (implicitly) that the mapping between t-variables and corresponding try-locks is a one-to-one relation. We discuss here how those assumptions can be relaxed (at the price of increasing the complexity of the definitions).

Arbitrary t-objects. Object-based TMs support t-objects of arbitrary type. However, most of them classify all the operations of t-objects as either read-only or update ones. In those cases, there is no need to consider arbitrary t-objects, because read-only operations are effectively reads, and update operations are effectively pairs of reads and writes.

We can, however, imagine a TM that exploits the commutativity relations between some operations of t-objects of any type. In this case, one can redefine the notion of a conflict to account for arbitrary t-objects. Indeed, operations that commute should not conflict. Consider for example a counter object and its operation inc that increments the counter and does not return any meaningful value. It is easy to see that there is no real conflict between transactions that concurrently invoke operation inc on the same counter: the order of those operations does not matter and is not known to transactions (it would be, however, if inc returned the current value of the counter). Once the notion of a conflict is redefined, our definitions of progress properties remain correct even for t-objects with arbitrary operations. If we assume that a TM must support t-variables (in addition to other t-objects), then also the consensus number and complexity lower bound results hold for those TMs.

False conflicts. Many lock-based TMs employ a hash function to map a t-variable (or, in general, a t-object) to the corresponding try-lock. It may thus happen that a false conflict occurs between transactions that access disjoint sets of t-variables, and so, a priori, strong progressiveness might be violated. However, it is easy to take the hash function $h$ of a TM implementation $M$ into account in the definition of strong progressiveness. Basically, when a transaction $T_i$ reads or writes a t-variable $x$ in a history $H$ of $M$, we add to, respectively, the read set ($\text{RSet}_h(T_i)$) or the write set ($\text{WSet}_h(T_i)$) of $T_i$ not only $x$, but also every t-variable $y$ such that $h(x) = h(y)$. The definition of a conflict hence also takes into account false conflicts between transactions, and the strong progressiveness property can be ensured by $M$. (Such a property could be called $h$-based strong progressiveness.) Note, however, that the hash function must be known to a user of a TM, or even provided by the user. Otherwise, strong progressiveness (and any property that relies on the notion of a conflict) would no longer be visible, and very meaningful, to a user.
6 Concluding Remarks

We gave an overview of our recent work on establishing the theoretical foundations of transactional memory (TM). We omitted many related results. We give here a short summary of some of those.

An important question is how to verify that a given history of a TM, or a given TM implementation, ensures opacity, obstruction-freedom, or strong progressiveness. In [20], we present a graph interpretation of opacity (similar in concept to the one of serializability [37, 5]). Basically, we show how to build a graph that represents the dependencies between transactions in a given history $H$. We then reduce the problem of checking whether $H$ is opaque to the problem of checking the acyclicity of this graph. In [21], we provide a simple reduction scheme that facilitates proving strong progressiveness of a given TM implementation $M$. Roughly speaking, we prove that if it is possible to say which parts of the algorithm of $M$ can be viewed as logical try-locks (in a precise sense we define in [21]), and if those logical try-locks are strong, then the TM is strongly progressive. In other words, if the locking mechanism used by $M$ is based on (logical) strong try-locks, then $M$ is strongly progressive.

The graph characterization of opacity and the reduction scheme for strong progressiveness do not address the problem of automatic model checking TM implementations. Basically, they do not deal with the issue of the unbounded number of states of a general TM implementation. In [16, 17], the problem is addressed for an interesting class of TMs. Basically, it is proved there that if a given TM implementation has certain symmetry properties, then it either violates opacity in some execution with only 2 processes and 2 t-variables, or ensures opacity in every execution (with any number of processes and t-variables). The theoretical framework presented in [16, 17] allows for automatic verifications of implementations such as DSTM or TL2 in a relatively short time.

One of the problems that we did not cover is the semantics of memory transactions from a programming language perspective. A very simple (but also very convenient) interface to a TM is via an `atomic` keyword that marks those blocks of code that should be executed inside transactions. The possible interactions between transactions themselves are confined by opacity. However, opacity does not specify the semantics of the interactions between the various programming language constructs that are inside and outside atomic blocks. Some work on those issues is presented, e.g., in [47, 31, 1, 36, 35].
The Bulletin of the EATCS

Acknowledgements

We thank Hagit Attiya, Aleksandar Dragojević, Pascal Felber, Christof Fetzer, Seth Gilbert, Vincent Gramoli, Tim Harris, Thomas Henzinger, Eshcar Hillel, Petr Kouznetsov, Leaf Petersen, Benjamin Pierce, Nir Shavit, Vasu Singh, and Jan Vitek for their helpful comments and discussions.

References


The Bulletin of the EATCS


Abstract

We are studying a particular graph abstraction, based on the concept of retaining neighbourhood information up to a certain radius for each node. In this contribution, we present a modal logic on graphs that is preserved and reflected by this abstraction. This is one of the main results from the EATCS Best Theoretical Paper at the International Conference on Graph Transformation 2008.
1 Introduction

Formal verification of graph transformation systems aims at statically proving or inferring properties of a graph transformation system, where such properties are typically given in some form of temporal logic. The main problem to be solved is the size of the state space, which often grows exponentially with the size of the system or may in fact be infinite. Moreover, even for finite state spaces, the individual states may be large and hence expensive to represent.

This work is a contribution to the concept of abstract graph transformation as introduced independently in [8, 9] and [1]. Abstract graph transformation relies on abstract interpretation [5] of graph transformation systems; that is, given some equivalence relation, graphs are quotiented into abstract graphs of finite, and in fact even bounded size. Application of graph transformation rules is then lifted to abstract graphs. Since the size of the individual graphs is bounded, the combined abstract state space is guaranteed to be finite, and can be drastically smaller than the concrete state space.

In this work, we generalise and unify the abovementioned approaches. To this end, we introduce the notion of neighbourhood abstraction. For this abstraction, nodes are summarised if they have similar neighbourhood up to some radius \( i \), which is a parameter of the abstraction. Different radii give rise to abstractions with different precisions. We also introduce a logic that precisely fits this type of abstraction: given a formula, our abstraction guarantees that a) if the formula holds for the original graph, then it holds for the abstracted graph (preservation); and b) if the formula holds for the abstracted graph, then it holds for the original one too (reflection).

Contributions. The full paper [2] contains the following results:

- Our abstraction framework unifies and generalises previous approaches on local graph abstractions. In contrast, equivalences used in shape analysis [11] of heap programs often consider global properties like reachability.
- Technically, the most surprising result is the definition of a modal logic, properties of which are preserved and reflected by our abstraction. While preservation is necessary for the soundness of analyses based on our abstraction, reflection is a rather unusual and strong result.
- Our framework allows automated abstraction refinement. If a property cannot be established given a certain neighbourhood size, one may automatically increase this size to obtain more precise results. The only other graph abstraction approach allowing abstraction refinement is [7].
A canonical representation of abstract graphs reduces otherwise costly isomorphism checks to simple equality tests.

Outline. In this extended abstract, we concentrate on contribution 2. In Section 2 we present the modal logic, and we show its utility on ordinary graphs. Section 3 defines our notion of graph abstraction and shows how to interpret the logic over abstract graphs. Finally, Section 4 presents the construction of the neighbourhood graphs and the preservation and reflection of the logic. We illustrate the theory on a running example, which models a simple internet firewall.

This paper is an excerpt of [2], which won the EATCS Best Theoretical Paper Award at the 2008 edition of the International Conference on Graph Transformation. Further discussion, a conclusion and a future work outlook can be found in the full paper.

2 A Modal Logic for Graphs

Preliminaries. We consider finite graphs with edges labelled from a finite set Lab of labels.

Definition 1 (graph). A graph G is a tuple \((N, E, \text{src}, \text{tgt}, \text{lab})\) where \(N\) is a finite set of nodes, \(E\) is a finite set of edges disjoint from \(N\), \(\text{src}, \text{tgt} : E \to N\) associate with each edge its source and target nodes, and \(\text{lab} : E \to \text{Lab}\) labels edges.

If the graph \(G\) is not clear from the context, we write \(N_G, E_G\) etc. for its components. A morphism between two graphs \(G, H\) is a pair of functions \(f_N : N_G \to N_H, f_E : E_G \to E_H\) which preserve the connections between edges, nodes and labels — concisely stated, \(f_N \circ \text{src}_G = \text{src}_H \circ f_E, f_N \circ \text{tgt}_G = \text{tgt}_H \circ f_E\) and \(\text{lab}_G = \text{lab}_H \circ f_E\).

In our examples, apart from edge labels we will also use node labels; these can be modelled by edges whose target is a special node \(\bot\). The set of labels of a node \(v\) is denoted \(\text{lab}(v)\). In the remainder of this extended abstract we will ignore this issue; for a formal treatment see the full paper, [2].

We introduce some auxiliary notation for the sets of outgoing and incoming edges for a given set of nodes, and the sets of edges between two sets of nodes:

\[
V \xrightarrow{>} a = \{ e \in E | \text{src}(e) \in V, \text{lab}(e) = a \}
\]

\[
V \xleftarrow{<} a = \{ e \in E | \text{tgt}(e) \in V, \text{lab}(e) = a \}
\]

\[
V \xrightarrow{\gg} W = X \xrightarrow{>} a \cap Y \xleftarrow{<} a.
\]

If \(V\) and \(W\) are singletons, we omit the set brackets; e.g., \(\{v\} \xrightarrow{>} a\) is denoted \(v \xrightarrow{>} a\).
As an example, consider Figure 1, which shows a graph model of two networks delimited by a firewall (FW). It has an internal and an external interface (IF), connected respectively to a network of in-locations (LI) to be protected by the firewall, and a network of out-locations (LO). Arbitrarily many packets flowing through the network can be created at locations – safe ones at any location and unsafe ones only at out-locations. The flow is bi-directional, despite the drawing of directed c-edges. The behaviour of such a firewalled system is described through a set of graph transformation rules, which are given in [3]. In this paper we will ignore the transformations and just concentrate on the graphs themselves.

A property we might want to verify for such a system is that unsafe packets never reach in-locations.

In our graph abstraction we will make heavy use of multiplicities, which count the number of elements in a given set up to a certain boundary; above the boundary, all values are combined into a single one. For any natural \( \mu > 0 \), we define \( M_\mu = \{0, 1, 2, \ldots, \mu, \omega\} \), where \( \omega \) is a special symbol representing “many” or “more than \( \mu \).” The multiplicity of a set \( X \) is defined by

\[
[X]_\mu = \begin{cases} 
0 & \text{if } |X| \leq \mu \\
\omega & \text{else}
\end{cases}
\]

Syntax and Semantics of the Logic. To express properties of graphs, we introduce a logic, \( L \). Formulae are defined by the following syntax (for \( a \in \text{Lab} \) and \( \lambda \in M_\mu \)):

\[
\phi ::= \text{tt} \mid a \mid \neg \phi \mid \phi \lor \phi \mid \langle a \rangle^\lambda \phi \mid \langle a \rangle^{\lambda'} \phi
\]

This is a modal logic, where \( \langle a \rangle^\lambda \) and \( \langle a \rangle^{\lambda'} \) are the modalities. These modalities express that a certain node has at least \( \lambda \) outgoing, respectively incoming, \( a \)-labelled edges, and the the opposite ends of each of those edges \( \phi \) holds. Apart from these modalities, the logic contains the usual negation and disjunction operators, as well as the proposition \( a \) which expresses that the node in question has a node label \( a \). Formally, the semantics is expressed by a satisfaction relation \( G, v \models \phi \), where \( G \) is a graph an \( v \in N \) a node of that graph, defined recursively on the structure of \( \phi \) by:

1. \( G, v \models \text{tt} \) always;
2. $G, v \models a$ if $a \in \text{lab}(v)$;
3. $G, v \models \neg \phi$ if $G, v \not\models \phi$;
4. $G, v \models \phi \lor \phi'$ if $G, v \models \phi$ or $G, v \models \phi'$;
5. $G, v \models \langle a \rangle^\lambda \cdot \phi$ if $\lambda \leq \|\{e \in v \triangleright a \mid G, \text{tgt}(e) \models \phi\}\|_\nu$;
6. $G, v \models \langle a \rangle^\lambda \cdot \phi$ if $\lambda \leq \|\{e \in v \triangleleft a \mid G, \text{src}(e) \models \phi\}\|_\mu$.

The nesting depth $d(\phi)$ measures the maximal number of nested modalities of the formula $\phi$. We will use fragments $\mathcal{L}_i$ of the full logic consisting only of formulae with maximal nesting depth $i$.

For instance, some properties for the firewall configuration of Fig. 1 are:

- $(\text{safe} \Rightarrow \neg \text{unsafe}) \land (\text{unsafe} \Rightarrow \neg \text{safe})$, expressing that a packet cannot be safe and unsafe simultaneously;
- $\neg (\text{LI} \land \langle c \rangle^1 \cdot \text{LO}) \land \neg (\text{LO} \land \langle c \rangle^1 \cdot \text{LI})$, expressing that an in-location cannot be directly connected to an out-location;
- $\neg (\text{LI} \land \langle \text{at} \langle 1 \cdot \text{unsafe} \rangle)$, expressing that no unsafe packet may appear on an in-location;
- $\text{Packet} \Rightarrow \neg \langle \text{at} \rangle^2 \cdot \text{tt}$, expressing that a packet has at most one current position.

All of these properties are in $\mathcal{L}_1$; moreover, the first one is in $\mathcal{L}_0$.

3 Abstract Graphs

We will use a notion of abstract graph, which is essentially a type graph with additional information. Recall that a type graph is in fact an ordinary graph, and a typing of a concrete graph into a type graph is a morphism between the two. Thus, a type graph is a very weak form of abstraction, which in fact only ensures the absence of certain edges.

The kind of information we add to this are multiplicities for both the nodes and the edges of the type graph, in line with the existing approaches of [11, 8]. The main innovation lies in the particular way we formalise the edge multiplicities: they are not attached to individual edges, but rather to groups of edges that are in a certain sense similar. The boundaries for the node and edge multiplicities can be different; in the definition below, $\nu$ is the boundary for the node multiplicities, and $\mu$ the boundary for the edge multiplicities.

**Definition 2** (abstract graph). An abstract graph $S$ is a tuple $(G, P, m_{\text{nod}}, m_{\text{out}}, m_{\text{in}})$, where...
The Bulletin of the EATCS

Figure 2: Examples of abstract graphs.

- $G$ is a graph;
- $P \subseteq 2^N$ is a partition of the nodes of $N$, called the grouping;
- $m_{nod} : N \to M_\nu^+$ is a node multiplicity function;
- $m_{out} : N \times \text{Lab} \times P \to M_\mu$ is an outgoing edges multiplicity function;
- $m_{in} : N \times \text{Lab} \times P \to M_\mu$ is an incoming edges multiplicity function.

Moreover, for any $v \in N$, $a \in \text{Lab}$ and $C \in P$, we require $m_{nod}(v, a, C) = 0$ iff $v \nsubseteq G_S C = \emptyset$, and $m_{in}(v, a, C) = 0$ iff $C \nsubseteq G_S v = \emptyset$.

The intuition behind the grouping is that the cells of the partition $P$, called the groups of $S$, are indistinguishable as far as the edge multiplicities from and to any of the nodes is concerned. Two example abstract graphs are shown in Figure 2. The dotted rectangles delimit the groups, and the edge multiplicities are written at the node from which they are calculated (which runs counter to the usual notation in UML).

Where the instances of an ordinary type graph need only to have a morphism to the type graph, obviously for our abstract graphs the requirements are stricter: the morphism has to ensure that the multiplicities are satisfied. This gives rise to the notion of an abstraction morphism; the graphs that have an abstraction morphism to a given abstract graph $S$ are called concretisations of $S$.

**Definition 3** (abstraction morphism). Let $G$ be a graph and $S$ be an abstract graph. An abstraction morphism of $G$ into $S$ is a graph morphism $s : G \to G_S$ such that the following conditions are met:

1. for all $w \in N_S$:
   \[ m_{nod}(w) = |s^{-1}(w)|_\nu; \]
2. for all $v \in N_G$, $C \in P_S$ and $a \in \text{Lab}$:
   \[ m_{nod}(s(v), a, C) = |v \nsubseteq G_S s^{-1}(C)|_\mu; \]
   \[ m_{in}(s(v), a, C) = |s^{-1}(C) \nsubseteq G_S v|_\mu. \]

For instance, the graphs in Fig. 3(a) are concretisations of Fig. 2(a), and similarly, the ones in Fig. 3(b) are concretisations of Fig. 2(b).
As another example, Fig. 4 below shows an abstract graph for the firewall example from Fig. 1, with $\nu = \mu = 1$. The corresponding abstraction morphism summarises the two LI-neighbours of the internal interface to the unique LI-neighbour of the internal interface in the abstract graph. The three LO nodes that have only LO neighbours are summarised to a unique node. All other nodes in the abstract graph have multiplicity one and correspond to a unique node in the concrete graph.

**Interpretation of the Logic over Abstract Graphs.** The satisfaction relation is defined for an abstract graph almost in the same way as for a (concrete) graph. The only difference lies in the rules for modalities: we no longer count individual edges, but sum up edge multiplicities instead. The following rules replace the ones in the original definition:

5. $S, v \models \Diamond a^1 \phi$ if $\lambda \leq \sum \{ m_{\text{out}}(v, a, C) \mid C \in P, \forall w \in C. S, w \models \phi \}$

6. $S, v \models (a^1 \phi$ if $\lambda \leq \sum \{ m_{\text{in}}(v, a, C) \mid C \in P, \forall w \in C. S, w \models \phi \}$

For instance, the abstract graph in Fig. 2(b) satisfies $b \Rightarrow (c^1 a$, meaning that every b-labelled node has at least one incoming c-edge from an a-labelled node.

However, without having more information about the relations $\equiv$ and $\sim$, we cannot say anything in general between the preservation of the logic $\mathcal{L}$ by abstraction. This is the topic of the next section.

### 4 Neighbourhood Abstraction

The definition of abstract graphs in Def. 2 is declarative and does not give a hint on their effective construction. We now present such a construction for a given graph $G$, based on two node equivalence relations $\equiv, \sim \subseteq N \times N$, such that $\equiv$ refines $\sim$. The idea is that $\equiv$-equivalent nodes are collapsed into a single node of the abstract graph, whereas $\sim$-equivalent nodes will be turned into groups.
For this to result in an abstraction of \( G \), \( \equiv \) should only equate nodes with the same local edge structure. To formalise this consistency requirement, we will call \( \equiv, \sim \) compatible with \( G \) if for all \( v \equiv w \) and all \( C \in N / \sim \), the following is satisfied for all \( a \in \text{Lab} \):

\[
\begin{align*}
|v \triangleright^a C|_p &= |w \triangleright^a C|_p \\
|C \triangleright^a v|_p &= |C \triangleright^a w|_p
\end{align*}
\]

Given compatible \( \equiv \) and \( \sim \), we define 

\[
G/\equiv, \sim = (G/\equiv, N/\equiv/\sim, m_{\text{nod}}, m_{\text{out}}, m_{\text{in}})
\]

where 

\[
\begin{align*}
m_{\text{nod}} &: V \mapsto |V|_p \\
m_{\text{out}} &: (V, a, C) \mapsto |v \triangleright^a C|_p \quad \text{for} \ v \in V \\
m_{\text{in}} &: (V, a, C) \mapsto |C \triangleright^a v|_p \quad \text{for} \ v \in V
\end{align*}
\]

Here \( G/\equiv \) is the quotient of \( G \), with nodes \([v]_\equiv\) for \( v \in N \), and \( N/\equiv/\sim \) is the partition consisting of cells \([w]_\equiv \mid w \sim v\) for \( v \in N \) (which is well-defined because \( \equiv \subseteq \sim \)). The multiplicity functions \( m_{\text{out}} \) and \( m_{\text{in}} \) are well-defined due to the compatibility condition above.

The following can now be shown:

**Proposition 4.** Given a graph \( G \) and compatible equivalences \( \equiv \) and \( \sim \), \( G/\equiv, \sim \) is an abstract graph, with abstraction morphism \( s : G \rightarrow S \) characterised by \( s(v) = [v]_\equiv \) for all \( v \in N \).

In fact, every abstract graph with a non-empty set of concretisations and without parallel edges can be constructed in this way.

**Neighbourhood Graphs.** We finally come to our proposed notion of **neighbourhood** abstraction of a graph, which is based upon a particular choice of the relations \( \equiv \) and \( \sim \) in the construction above. This gives rise to so-called **neighbourhood graphs**, in which each node represents concrete graph nodes that have similar neighbourhood, up to some “radius” \( i \). This \( i \) is a parameter of the precision of the neighbourhood abstraction. By gradually increasing \( i \), we can obtain more precise abstractions, if the current one is too imprecise to verify the desired properties (abstraction refinement). We shall now define equivalence between nodes according to their neighbourhood and, subsequently, neighbourhood abstraction.

**Definition 5** (neighbourhood equivalence). Let \( G \) be a graph. For each \( i \geq 0 \), we define the \( i \)-neighbourhood equivalence relation \( \equiv_i \) over \( N_G \) recursively by:

\[\text{The complete formalisation and proof of this construction can be found in [4].}\]
Figure 4: The 1-neighbourhood graph of Fig. 1. Omitted node and edge multiplicities are equal to 1.

- $v \equiv_0 w$ if $\text{lab}(v) = \text{lab}(w)$;
- $v \equiv_{i+1} w$ if (i) $v \equiv_i w$, and (ii) $|v \triangleright^a C|_\mu = |w \triangleright^a C|_\mu$, and $|C \triangleright^a v|_\mu = |C \triangleright^a w|_\mu$ for all label $a \in \text{Lab}$ and for all $C \in N / \equiv_i$.

The following is straightforward to show.

**Proposition 6.** For all $i > 0$, $\equiv_i$, $\equiv_{i-1}$ are compatible with $G$.

For $i > 0$, the $i$-neighbourhood graph of $G$ is given by $G/\equiv_i, \equiv_{i-1}$. Thus, concrete nodes are summarised (i.e., mapped to the same abstract node) if they are $i$-neighbourhood equivalent up to $i$, whereas the groups are formed by $i-1$-neighbourhood equivalence.

For instance, Fig. 4 shows the level 1 neighbourhood graph of the firewall configuration from Fig. 1 for $\mu = 1$ and $\nu = 1$.

It is obvious from the definition that the $i+1$-neighbourhood graph refines the $i$-neighbourhood graph. This is the basis of our abstraction refinement mechanism.

**Preservation and Reflection.** We now come to the main result of this paper. Let $s : G \to S$ be an abstraction morphism from the graph $G$ to the abstract graph $S$. We say that $s$ preserves a property $p \in \mathcal{L}$ if whenever $p$ holds in the node $v$ of $G$, it also holds in the node $s(v)$ of $S$. Inversely, we say that $s$ reflects $p$ if whenever $p$ holds in the node $s(v)$ of $S$, it also holds in the node $v$ of $G$. One can also define in a similar manner preservation and reflection by an abstraction morphism between abstract graphs.

Preservation and reflection are very important characterisations. If an abstraction preserves a set of safety properties, these properties can be verified on the abstract level. If an abstraction reflects a set of properties, then any characterisation of an abstract graph also holds for its concretisations. If both preservation and reflection hold, verifying a property on a graph is equivalent to verifying it on the abstract level. The following theorem, which is one of the main results of this work, states that neighbourhood abstraction guarantees preservation and reflection of logic formulae with appropriate nesting depth, as stated in the following
Theorem 7 (Preservation and reflection). Let \( G \) be a graph and \( S \) the level \( i \) neighbourhood abstraction of \( G \), for some \( i \geq 1 \), with corresponding abstraction morphism \( s : G \to S \). Then \( s \) preserves and reflects \( L_i \).

An important consequence of it is that neighbourhood abstraction can be parametrised by the properties we want to verify by choosing the level of abstraction that preserves the properties one is interested in.

For instance, by observing that the abstract graph in Fig. 4 satisfies the property \( \neg (L_1 \land \langle \text{at}^{(1)} \cdot \text{unsafe}\rangle) \) discussed in Section 2, we know that all its concretisations also satisfy this property; but also, if by transforming this abstract graph (which is something not discussed in this paper) we would obtain a new abstract graph, say \( S' \), where this property is not fulfilled, then none of the concretisations of \( S' \) would fulfill this property.

In the proof of Theorem 7, the following intermediate result is interesting in its own right. It turns out that \( L_i \) is essentially a logical characterisation of \( \equiv_i \). This is entirely analogous to (and an extension of) the well-known characterisation of bisimilarity by Hennessy-Milner logic (see [6]). Formally stated, the property is the following: equivalence:

**Proposition 8.** Two nodes \( v, v' \) of a graph \( G \) are \( \equiv_i \)-equivalent if, and only if, the same \( L_i \)-formulae hold in \( v \) and in \( v' \).

For our running example, let \( G \) be the graph of Fig. 1 and \( S \) its level 1 neighbourhood abstraction of Fig. 4. Let \( s : G \to S \) be the corresponding abstraction morphism, and let \( \phi = L_0 \land \langle \text{c} \rangle^{(1)} \cdot \text{IF} \) and \( \psi = L_0 \land \langle \text{c} \rangle^{(1)} \cdot \langle \text{c} \rangle^{(1)} \cdot \text{IF} \). In \( G \), \( \phi \) only holds for the \( L_0 \)-neighbour of the out interface, and in \( S \), \( \phi \) only holds for the corresponding abstract node. That is, \( \phi \) of nesting depth 1 is preserved and reflected by \( s \), whereas \( \psi \), a formula of nesting depth 2, is not reflected. Indeed, in \( S \), \( \phi \) holds in the \( L_0 \)-node with multiplicity \( \omega \) but only in one of the pre-images of this node in \( G \).

\[ \text{Preservation of formulae with negation may seem in contradiction with the Morphism Preservation Theorem for finite structures [10] stating that a first order formula is preserved by morphism iff it is equivalent to an existential positive formula. Some modal logic formulae cannot be expressed in first-order logic without negation (e.g. } \neg \langle a \rangle^{(1)} \cdot \text{If}). \] However, in our case, abstract graphs contain information on the interpretation of negated formulae, by means of the multiplicity functions explaining this apparent contradiction.
References


Behavior Simulation and Equivalence of Systems Modelled by Graph Transformation

Hartmut Ehrig∗, Claudia Ermel∗, Frank Hermann∗
Barbara König †

Abstract

Our new project "Behavior Simulation and Equivalence of Systems Modelled by Graph Transformation" funded by the German Research Council (DFG) has started in May 2008. In this contribution we present the main goals of the project and first results.

1 Introduction

For about 35 years, various specification techniques have been developed in order to model different kinds of systems in computer science and other areas. The most straightforward techniques are based on automata and transition systems. In order to model the behavioural aspects of systems including their concurrent behaviour especially Petri nets [25] and process algebras like CCS [22] and CSP [18] have become prominent. More recently, object-oriented and visual modelling techniques have been used frequently in the software development process, where especially UML [23] has become a quasi-standard for industrial software development. It is well-known, however, that there is no standard formal semantics for UML, which is a problem for the formal analysis and verification of system behaviour. For this reason UML models are often transformed into formal models in order to support analysis techniques. A large variety of UML diagrams and other visual models can be represented on the level of abstract syntax by suitable graphs such that the operational semantics can be modelled by graph transformation (see e.g. [1, 29, 20, 17]). Moreover, graph transformation systems are suitable to define model transformations on a formal basis [4, 30] and there is a close relationship with process algebras.

∗Fakultät für Elektrotechnik und Informatik, Technische Universität Berlin, Germany, ehrig|lieske|frank(at)cs.tu-berlin.de
†Abteilung für Informatik und Angewandte Kognitionswissenschaft, Universität Duisburg-Essen, Germany, barbara_koenig@uni-due.de

Bulletin of the EATCS no 97, pp. 117–125, February 2009
© European Association for Theoretical Computer Science
The algebraic graph transformation approach was initiated by H. Ehrig and M. Pfender at TU Berlin in cooperation with H. J. Schneider in Erlangen [15]. It is based on two gluing constructions for graphs, which are pushouts in the category of graphs. For this reason, the approach is also known as the double-pushout (DPO) approach. The first major step in the theory was the local Church-Rosser Theorem for graph transformations developed in cooperation with B. K. Rosen from IBM-Yorktown Heights [16], which is an important step towards local confluence. The theory was further developed mainly in cooperation of TU Berlin with H. J. Kreowski (Berlin/Bremen), U. Montanari and A. Corradini (Pisa), F. Parisi Presicce (USC/Rome), M. Löwe (Berlin/Hannover), R. Heckel (Berlin/Paderborn/Leicester) and G. Taentzer (Berlin/Marburg) motivated by the transfer of concepts from Petri nets to graph grammars and applications to software development as well as to visual design and modelling (see Handbooks on Graph Transformation [28, 9, 13]. Main results concerning graph transformation for operational semantics are the local Church-Rosser theorem, the embedding and extension theorem, the completeness of critical pairs and the critical pair lemma leading to local confluence. Together with termination this leads to confluence of graph transformation systems and hence, to unique normal forms for the operational semantics (see [10]).

Motivated by Milner’s bigraph approach the DPO-approach was extended in view of process algebra and interaction with the environment by H. Ehrig and B. König in [11, 12], called DPO-approach with borrowed context. This approach allows us to apply rules with partial matches by borrowing context from the environment and to transfer the concept of behavioural equivalences from process algebra to graph rewriting. Especially, it introduces a simple technique for the derivation of interaction labels [11, 12]. This technique has then been used in order to show the equivalence of the original CCS bisimilarity and a derived bisimulation equivalence in [3]. Furthermore, first steps towards a structural operational semantics (SOS semantics) for graph transformation systems are given in [2], i.e., towards a framework where the transition system associated to a graph transformation system can be defined inductively. This modular view – quite common in process algebra – is important for the comparison of formalisms and modular proof techniques. Finally, different variants of bisimilarity, especially saturated semantics, are studied in [BKM06]. Earlier work in this direction was concerned with the definition of notions of behavioural equivalence for synchronized hyperedge replacement [19]. A Dagstuhl seminar entitled “Graph Transformations and Process Algebras for Modelling Distributed and Mobile Systems” organized by Barbara König together with Ugo Montanari and Philippa Gardner (Imperial College, London) took place in June 2004 in order to promote further collaboration in this area. This was continued by the GT-VC workshop (“Graph Transformation for Verification and Concurrency”) as satellite workshop of CONCUR in 2005-
The Bulletin of the EATCS

2007.

In the following Sections 2 and 3 we present the motivation, goals and first results of a new DFG-project “Behaviour Simulation and Equivalence of Systems Modelled by Graph Transformation”, which is based on the techniques discussed above. A summary of the project is given in Section 4.

2 Motivation and Goals of the Project

Graph transformation has been very successful to model systems in all kinds of application areas, especially concurrent, distributed and dynamically evolving systems. The graphical notation is especially useful to model states, state transformation and hence, the behaviour of systems. We are especially interested in the application area of model-driven architectures which requires the construction of several models and their conversion into one another during the software development process. This transformation of models should satisfy suitable correctness criteria, such as termination, confluence and preservation of behaviour. Unfortunately these aspects are not well-supported by the existing theory up to now. One of the main goals of this project is to show how these aspects can be supported by the theory of graph transformation systems extended by suitable notions of behaviour simulation and equivalence.

In order to study the behaviour of graph transformation systems, it is first necessary to define an operational semantics, faithfully modelling all feasible transitions. Based on this semantics it is then possible to define various levels of abstraction and views or to answer questions concerning confluence and termination. The semantics can either be closed, i.e., it assumes that the system is fully specified, or open, i.e., it takes unspecified interactions with the environment into account. Especially the open or interactive view is needed in order to identify systems which are considered to be behaviourally equivalent from the point of view of an external observer.

Operational semantics and behavioural equivalences are closely related issues. Changing the operational semantics implies changing the notion of equivalence. There are many interesting questions related to this aspect: What are the possible views on a system and how do they change the observational power of an observer, leading to a change in the granularity of behavioural equivalence? How are confluence, termination and determinism related to notions of equivalence?

Concerning both, behaviour simulation and equivalence, there are mainly two approaches in the theory of (algebraic) graph transformation:

1. transformation systems in order to model operational semantics and hence, simulation of behaviour in the sense of rewriting systems,
2. transformations with borrowed context inspired by process algebra and Milner’s bigraphs, which allow to study behavioural equivalences congruences in a systematic way.

The first main goal of this project is to develop the behavioural aspects of both approaches systematically. This includes especially how to transfer concepts between process algebra, graph transformations with borrowed context and other graph transformation approaches in order to develop feasible techniques for behaviour simulation and equivalence. Moreover, we will use the general framework of adhesive categories as far as possible.

The second main goal is to use these results in the application areas of model transformation, including the important special case of model refactoring, and protocol verification. This will especially allow to study the problems of semantical correctness for model refactoring and transformation based on graph transformation. Last but not least, tool support will be provided to calculate properties concerning behaviour simulation and equivalence and the result will be evaluated by suitable case studies in the corresponding application areas. Although our applications come from quite diverse areas, they need a common underlying theory, which will be provided in the more foundational work packages.

In order to achieve these goals we have structured the work schedule into the following parts:

1. Behaviour Simulation and Operational Semantics
2. Behavioural Equivalence and Process Algebras
3. General Framework and Different Approaches
4. Application Areas
5. Case Studies and Tool Support

Parts 1-3 concern more foundational research on behaviour simulation and equivalence, with an emphasis to relations and ties with rewriting systems and process algebra, whereas part 4 focuses on our application areas. Finally, part 5 includes case studies and tool development. Schematically this is represented in the figure below.
3 First Results of the Project

Concerning the general topic "Behaviour Simulation and Equivalence of Systems Modelled by Graph Transformation" and especially Part 1 above, we have shown in [21] how to extend main results for embedding and confluence of graph transformation in [10] to the case with negative application conditions. Application conditions in general are important for consistent modelling of visual languages and their operational semantics. In [6] we have given sufficient conditions for consistent integration of models based on views of visual languages.

The double pushout approach with borrowed context introduced in [12] has been extended to rules with negative application conditions in [26] and in [27] it has been applied to show behaviour preservation in model refactoring.

Results of the project shall be formulated as far as possible on the general framework of weak adhesive HLR categories. They are closed under certain constructions, e.g. product, slice, coslice, comma and functor category construction. However, special additional properties have to be valid as well in the constructed categories in order to make available the full theory. In [24] we have analyzed under which conditions the additional properties are preserved by the categorical constructions to avoid checking these properties explicitly.
Model transformation is one application domain of the project and there are several specializations, e.g. model refactoring. In [7] we have shown how model integration fits into the concept of model transformation based on triple graph grammars. In particular, there is a unique model integration sequence for each model transformation sequence and vice versa. In order to relate existing implementations and case studies of model transformations based on plain graph grammars to the framework of triple graph grammars, we have shown a one-to-one correspondence between triple graph grammars and suitable plain graph grammars in [8]. Thus results and benefits of the triple case can be transferred to the plain case.

Model transformation shall furthermore show functional behaviour and termination. These properties are difficult to check for plain graph grammars as well as for triple graph grammars in general. In [14] we have analyzed the relationship of the different conditions, which ensure the properties above. In particular, we analyzed how source consistency of forward transformations in the triple case is related to NAC consistency and termination in other model transformation approaches. We defined sufficient conditions such that source consistency implies NAC consistency and termination. Moreover, we analyzed how to achieve local confluence independent of source consistency.

An important requirement of model transformations is behaviour preservation of the original model. In [5] we use simulation rules in order to specify the behaviour of models and show conditions for model and rule transformations to be semantically correct and complete. From an observational point of view the behaviour of a model can be defined using DPO with borrowed context and in this case we have shown in [27] how to check refactoring rules for behaviour preservation.

4 Summary

The current trend to model driven software and system development requires the construction of different kinds of models and model transformations. In order to validate such transformations, behaviour preservation and refinement are important, but often neglected issues. In this project we concentrate on graph transformation as a modelling language, due to its success in modelling dynamically evolving graphical structures and system architectures. Up to now, however, there is no systematic study of behaviour simulation and equivalence for graph transformation systems. The main aim of this project is to fill this gap and to apply the corresponding results and techniques to the problem of behaviour preservation of model transformations. For this purpose we transfer on the one hand concepts of behaviour simulation from operational semantics defined by rewriting systems and
on the other hand concepts of behavioural equivalences from the area of process algebras to the algebraic theory of graph transformations. In addition to model transformation in general the results will be applied to the special case of model refactoring and to protocol verification. Moreover, tool support will be provided for behaviour simulation and equivalence and the results will be evaluated in several case studies.

References


The Bulletin of the EATCS


THE LOGIC IN COMPUTER SCIENCE COLUMN

BY

YURI GUIREVICH

Foundations of Software Engineering
Microsoft Research
One Microsoft Way, Redmond WA 98052, USA
gurevich@microsoft.com

FIXED POINT THEOREM AND APERIODIC TILINGS

Bruno Durand, Andrei Romashchenko and Alexander Shen*

Abstract

A new simple construction of an aperiodic tile set based on self-referential (fixed point) argument is proposed.

People often say about some discovery that it appeared “ahead of time”, meaning that it could be fully understood only in the context of ideas developed later. For the topic of this note, the construction of an aperiodic tile set based on the fixed-point (self-referential) approach, the situation is exactly the opposite. It should have been found in 1960s when the question about aperiodic tile sets was first asked: all the tools were quite standard and widely used at that time. However, the history had chosen a different path and many nice geometric ad hoc constructions were developed instead (by Berger, Robinson, Penrose, Ammann and many others, see [6]; a popular exposition of Robinson-style construction is given in [3]). In this note we try to correct this error and present a construction that should have been discovered first but seemed to be unnoticed for more that forty years.

*LIF Marseille, CNRS & University Aix-Marseille. Partially supported by ANR (Sycomore and Naft grants) and RFBR (05-01-02803, 06-01-00122a), IITP RAS.
1 The statement: aperiodic tile sets

A tile is a square with colored sides. Given a set of tiles, we want to find a tiling, i.e., to cover the plane by (translated copies of) these tiles in such a way that colors match (a common side of two neighbor tiles has the same color in both).

For example, if tile set consists of two tiles (one has black lower and left side and white right and top sides, the other has the opposite colors), it is easy to see that only periodic (checkerboard) tiling is possible. However, if we add some other tiles the resulting tile set may admit also non-periodic tilings (e.g., if we add all 16 possible tiles, any combination of edge colors becomes possible). It turns out that there are other tile set that have only aperiodic tilings.

Formally: let $C$ be a finite set of colors and let $\tau \subset C^4$ be a set of tiles; the components of the quadruple are interpreted as upper/right/lower/left colors of a tile. Our example tile set with two tiles is represented then as

$$\{\langle \text{white}, \text{white}, \text{black}, \text{black} \rangle, \langle \text{black}, \text{black}, \text{white}, \text{white} \rangle\}.$$  

A $\tau$-tiling is a mapping $\mathbb{Z}^2 \rightarrow \tau$ that satisfies matching conditions. Tiling $U$ is called periodic if it has a period, i.e., if there exists a non-zero vector $T \in \mathbb{Z}^2$ such that $U(x + T) = U(x)$ for all $x$.

Now we can formulate the result (first proven by Berger [1]):

1 Tiles appeared first in the context of domino problem posed by Hao Wang. Here is the original formulation from [10]: "Assume we are given a finite set of square plates of the same size with edges colored, each in a different manner. Suppose further there are infinitely many copies of each plate (plate type). We are not permitted to rotate or reflect a plate. The question is to find an effective procedure by which we can decide, for each given finite set of plates, whether we can cover up the whole plane (or, equivalently, an infinite quadrant thereof) with copies of the plates subject to the restriction that adjoining edges must have the same color." This question (domino problem) is closely related to the existence of aperiodic tile sets: (1) if they did not exist, domino problem would be decidable for some simple reasons (one may look in parallel for a periodic tiling or a finite region that cannot be tiled) and (2) the aperiodic tile sets are used in the proof of the undecidability of domino problem. However, in this note we concentrate on aperiodic tile sets only.
Proposition. There exists a finite tile set $\tau$ such that $\tau$-tilings exist but all of them are aperiodic.

There is a useful reformulation of this result. Instead of tilings we can consider two-dimensional infinite words in some finite alphabet $A$ (i.e., mappings of type $\mathbb{Z}^2 \to A$) and put some local constraints on them. This means that we choose some positive integer $N$ and look at the word through a window of size $N \times N$. Local constraint then says which patterns of size $N \times N$ are allowed to appear in a window. Now we can reformulate our Proposition as follows: there exists a local constraint that is consistent (some infinite words satisfy it) but implies aperiodicity (all satisfying words are aperiodic).

It is easy to see that these two formulations are equivalent. Indeed, the color matching condition is $2 \times 2$ checkable. On the other hand, any local constraint can be expressed in terms of tiles and colors if we use $N \times N$-patterns as tiles and $(N-1) \times N$-patterns as colors; e.g., the right color of $(N \times N)$-tile is the tile except for its left column; if it matches the left color of the right neighbor, these two tiles overlap correctly.

2 Why theory of computation?

At first glance this proposition has nothing to do with theory of computation. However, the question appeared in the context of the undecidability of some logical decision problems, and, as we shall see, can be solved using theory of computations. (A rare chance to convince “normal” mathematicians that theory of computations is useful!)

The reason why theory of computation comes into play is that rules that determine the behavior of a computation device — say, a Turing machine with one-dimensional tape — can be transformed into local constraints for the space-time diagram that represents computation process. So we can try to prove the proposition as follows: consider a Turing machine with a very complicated (and therefore aperiodic) behavior and translate its rules into local constraints; then any tiling represents a time-space diagram of a computation and therefore is aperiodic.

However, this naïve approach does not work since local constraints are satisfied also at the places where no computation happens (in the regions that do not contain the head of a Turing machine) and therefore allow periodic configurations. So a more sophisticated approach is needed.
3 Self-similarity

The main idea of this more sophisticated approach is to construct a “self-similar” set of tiles. Informally speaking, this means that any tiling can be uniquely split by vertical and horizontal lines into $M \times M$ blocks that behave exactly like the individual tiles. Then, if we see a tiling and zoom out with scale $1 : M$, we get a tiling with the same tile set.

Let us give a formal definition. Assume that a non-empty set of tiles $\tau$ and positive integer $M > 1$ are fixed. A macro-tile is a square of size $M \times M$ filled with matching tiles from $\tau$. Let $\rho$ be a non-empty set of macro-tiles.

**Definition.** We say that $\tau$ implements $\rho$ if any $\tau$-tiling can be uniquely split by horizontal and vertical lines into macro-tiles from $\rho$.

Now we give two examples that illustrate this definition: one negative and one positive.

**Negative example:** Consider a set $\tau$ that consists of one tile with all white sides. Then there is only one macro-tile (of given size $M \times M$). Let $\rho$ be a one-element set that consists of this macro-tile. Any $\tau$-tiling (i.e., the only possible $\tau$-tiling) can be split into $\rho$-macro-tiles. However, the splitting lines are not unique, so $\tau$ does not implement $\rho$.

**Positive example:** Let $\tau$ is a set of $M^2$ tiles that are indexed by pairs of integers modulo $M$: The colors are pairs of integers modulo $M$ arranged as shown

$$
\begin{array}{c}
(i, j + 1) \\
(i, j) \boxed{, (i + 1, j)} \\
(i, j)
\end{array}
$$

(Fig. 2). Then there exists only one $\tau$-tiling (up to translations), and this tiling can be uniquely split into $M \times M$ squares whose borders have colors $(0, j)$ and $(i, 0)$. Therefore, $\tau$ implements a set $\rho$ that consists of one macro-tile (Fig. 3).

**Definition.** A set of tiles $\tau$ is self-similar if it implements some set of macro-tiles $\rho$ that is isomorphic to $\tau$.

This means that there exist a 1-1-correspondence between $\tau$ and $\rho$ such that matching pairs of $\tau$-tiles correspond exactly to matching pairs of $\rho$-macro-tiles.

The following statement follows directly from the definition:

**Proposition.** A self-similar tile set $\tau$ has only aperiodic tilings.
Figure 3: The only element of $\rho$: border colors are pairs that contain 0

**Proof.** Let $T$ be a period of some $\tau$-tiling $U$. By definition $U$ can be uniquely split into $\rho$-macro-tiles. Shift by $T$ should respect this splitting (otherwise we get a different splitting), so $T$ is a multiple of $M$. Zooming the tiling and replacing each $\rho$-macro-tile by a corresponding $\tau$-tile, we get a $T/M$-shift of a $\tau$-tiling. For the same reason $T/M$ should be a multiple of $M$, then we zoom out again etc. We conclude therefore that $T$ is a multiple of $M^k$ for any $k$, i.e., $T$ is a zero vector.

Note also that any self-similar set $\tau$ has at least one tiling. Indeed, by definition we can tile a $M \times M$ square (since macro-tiles exist). Replacing each $\tau$-tile by a corresponding macro-tile, we get a $\tau$-tiling of $M^2 \times M^2$ square, etc. In this way we can tile an arbitrarily large finite region, and then standard compactness argument (König’s lemma) shows that we can tile the entire plane.

So it remains to construct a self-similar set of tiles (a set of tiles that implements itself, up to an isomorphism).

## 4 Fixed points and self-referential constructions

The construction of a self-similar tile set is done in two steps. First (in Section 5) we explain how to construct (for a given tile set $\sigma$) another tile set $\tau$ that implements $\sigma$ (i.e., implements a set of macro-tiles isomorphic to $\sigma$). In this construction the tile set $\sigma$ is given as a program $p_\sigma$ that checks whether four bit strings (representing four side colors) appear in one $\sigma$-tile. The tile set $\tau$ then guarantees that each macro-tile encodes a computation where $p_\sigma$ is applied to these four strings (“macro-colors”) and accepts them.

This gives us a mapping: for every $\sigma$ we have $\tau = \tau(\sigma)$ that implements $\sigma$ and depends on $\sigma$. Now we need a fixed point of this mapping where $\tau(\sigma)$ is isomorphic to $\sigma$. It is done (Section 6) by a classical self-referential trick that appeared as liar’s paradox, Cantor’s diagonal argument, Russell’s paradox, Gödel’s (first) incompleteness theorem, Tarsky’s theorem, undecidability of the Halting problem, Kleene’s fixed point (recursion) theorem and von Neumann’s construction of
The Bulletin of the EATCS

self-reproducing automata — in all these cases the core argument is essentially the same.

The same trick is used also in a classical programming challenge: to write a program that prints its own text. Of course, for every string $s$ it is trivial to write a program $t(s)$ that prints $s$, but how do we get $t(s) = s$? It seems at first that $t(s)$ should incorporate the string $s$ itself plus some overhead, so how $t(s)$ can be equal to $s$? However, this first impression is false. Imagine that our computational device is a universal Turing machine $U$ where the program is written in a special read-only layer of the tape. (This means that the tape alphabet is a Cartesian product of two components, and one of the components is used for the program and is never changed by $U$.) Then the program can get access to its own text at any moment, and, in particular, can copy it to the output tape.

Now we explain in more details how to get a self-similar tile set according to this scheme.

5 Implementing a given tile set

In this section we show how one can implement a given tile set $\sigma$, or, better to say, how to construct a tile set $\tau$ that implements some set of macro-tiles that is isomorphic to $\sigma$.

There are easy ways to do this. Though we cannot let $\tau = \sigma$ (recall that zoom factor $M$ should be greater than 1), we can do essentially the same for every $M > 1$. Let us extend our “positive” example (with one macro-tile and $M^2$ tiles) by superimposing additional colors. Superimposing two sets of colors means the we consider the Cartesian product of color sets (so each edge carries a pair of colors). One set of colors remains the same ($M^2$ colors for $M^2$ pairs of integers modulo $M$). Let us describe additional (superimposed) colors. Internal edges of each macro-tile should have the same color and this color should be different for all macro-tiles, so we allocate $#\sigma$ colors for that. This gives $#\sigma$ macro-tiles that can be put into 1-1-correspondence with $\sigma$-tiles. It remains to provide correct border colors, and this is easy to do since each tile “knows” which $\sigma$-tile it simulates (due to the internal color). In this way we get $M^2#\sigma$ tiles that implement the tile set $\sigma$ with zoom factor $M$.

However, this (trivial) simulation is not really useful. Recall that our goal is

\[10\] \texttt{LIST}

that indeed prints its own text. However, this trick can be generalized enough to show that a self-printing program exists in every language.
to get isomorphic \( \sigma \) and \( \tau \), and in this implementation \( \tau \)-tiles have more colors that \( \sigma \)-tiles (and we have more tiles, too). So we need a more creative encoding of \( \sigma \)-colors that makes use of the space available: a side of a macro-tile has a “macro-color” that is a sequence of \( M \) tile colors, and we can have a lot of macro-colors in this way.

So let us assume that colors in \( \sigma \) are \( k \)-bit strings for some \( k \). Then the tile set is a subset \( S \subset \mathbb{B}^k \times \mathbb{B}^k \times \mathbb{B}^k \times \mathbb{B}^k \), i.e., a 4-ary predicate on the set \( \mathbb{B}^k \) of \( k \)-bit strings. Assume that \( S \) is presented by a program that computes Boolean value \( S(x, y, z, w) \) given four \( k \)-bit strings \( x, y, z, w \). Then we can construct a tile set \( \tau \) as follows.

We start again with a set of \( M^2 \) tiles from our example and superimpose additional colors but use them in a more economical way. Assuming that \( k \ll M \), we allocate \( k \) places in the middle of each side of a macro-tile and allow each of them to carry an additional color bit; then a macro-color represents a \( k \)-bit string. Then we need to arrange the internal colors in such a way that macro-colors (\( k \)-bit strings) \( x, y, z \) and \( w \) can appear on the four sides of a macro-tile if and only if \( S(x, y, z, w) \) is true.

To achieve this goal, let us agree that the middle part (of size, say, \( M/2 \times M/2 \)) in every \( M \times M \)-macro-tile is a “computation zone”. Tiling rules (for superimposed colors) in this zone guarantee that it represents a time-space diagram of a computation of some (fixed) universal Turing machine. (We assume that time goes up in a vertical direction and the tape is horizontal.) It is convenient to assume that program of this machine is written on a special read-only layer of the tape (see the discussion in Section 4).

Outside the computation zone the tiling rules guarantee that bits are transmitted from the sides to the initial configuration of a computation.

We also require that this machine should accept its input before running out of time (i.e., less than in \( M/2 \) steps), otherwise the tiling is impossible.

Note that in this description different parts of a macro-tile behave differently; this is OK since we start from our example where each tile “knows” its position in a macro-tile (keeps two integers modulo \( M \)). So the tiles in the “wire” zone know that they should transmit a bit, the tiles inside the computation zone know they should obey the local rules for time-space diagram of the computation, etc.

This construction uses only bounded number of additional colors since we have fixed the universal Turing machine (including its alphabet and number of states); we do not need to increase the number of colors when we increase \( M \) and \( k \) (though \( k \) should be small compared to \( M \) to leave enough space for the wires; we do not give an exact position of the wires but it is easy to see that if \( k/M \) is small enough, there is enough space for them). So the construction uses \( O(M^2) \) colors (and tiles).
6 A tile set that implements itself

Now we come to the crucial point in our argument: can we arrange things in such a way that the predicate $S$ (i.e., the tile set it generates) is isomorphic to the set of tiles $\tau$ used to implement it?

Assume that $k = 2 \log M + O(1)$; then macro-colors have enough space to encode the coordinates modulo $M$ plus superimposed colors (which require $O(1)$ bits for encoding).

Note that many of the rules that define $\tau$ do not depend on $\sigma$ (i.e., on the predicate $S$). So the program for the universal Turing machine may start by checking these rules. It should check that

- bits that represent coordinates (integers modulo $M$) on the four sides of a macro-tile are related in the proper way (left and lower sides have identical coordinates, on the right/upper side one of the coordinates increases modulo $M$);
- if the macro-tile is outside computation zone and the wires, it does not carry additional colors;
- if the macro-tile is a part of a wire, then it transmits a bit in a required direction (of course, for this we should fix the position of the wires by some formulas that are then checked by a program);
• if the macro-tile is a part of the computation zone, it should obey the local rules for the computation zone (bits of the read-only layer should propagate vertically, bits that encode the content of the tape and the head of our universal Turing machine should change as time increases according to the behavior of this machine, etc.)

This guarantees that on the next layer macro-tiles are grouped into macro-macro-tiles where bits are transmitted correctly to the computation zone of a macro-macro-tile and some computation of the universal Turing machine is performed in this zone. But we need more: this computation should be the same computation that is performed on the macro-tile level (fixed point!). This is also easy to achieve since in our model the text of a running program is available to it (recall the we assume that the program is written in a read-only layer): the program should check also that if a macro-tile is in the computation zone, then the program bit it carries is correct (program knows the x-coordinate of a macro-tile, so it can go at the corresponding place of its own tape to find out which program bit resides in this place).

This sound like some magic, but we hope that our previous example (a program for the UTM that prints its own text) makes this trick less magical (indeed, reliable and reusable magic is called technology).

7 So what?

We believe that our proof is rather natural. If von Neumann lived few years more and were asked about aperiodic tile sets, he would probably immediately give this argument as a solution. (He was especially well prepared to it since he used very similar self-referential tricks to construct a self-reproducing automata, see [9].) In fact this proof somehow appeared, though not very explicitly, in P. Gács’ papers on cellular automata [5]; the attempts to understand these papers were our starting points.

This proof is rather flexible and can be adapted to get many results usually associated with aperiodic tilings: undecidability of domino problem (Berger [1]), recursive inseparability of periodic tile sets and inconsistent tile sets (Gurevich – Koryakov [7]), enforcing substitution rules (Mozes [8]) and others (see [2, 4]). But does it give something new?

We believe that indeed there are some applications that hardly could be achieved by previous arguments. Let us conclude by mentioning two of them. First is the construction of robust aperiodic tile sets. We can consider tilings with holes (where no tiles are placed and therefore no matching rules are checked). A robust aperiodic tile set should have the following property: if the set of holes is
“sparse enough”, then tiling still should be far from any periodic pattern (say, in the sense of Besicovitch distance, i.e., the limsup of the fraction of mismatched positions in a centered square as the size of the square goes to infinity). The notion of "sparsity" should not be too restrictive here; we guarantee, for example, that a Bernoulli random set with small enough probability $p$ (each cell belongs to a hole independently with probability $p$) is sparse.

While the first example (robust aperiodic tile sets) is rather technical (see [4] for details), the second is more basic. Let us split all tiles in some tile set into two classes, say, A- and B-tiles. Then we consider a fraction of A-tiles in a tiling. If a tile set is not restrictive (allows many tilings), this fraction could vary from one tiling to another. For classical aperiodic tilings this fraction is usually fixed: in a big tiled region the fraction of A-tiles is close to some limit value, usually an eigenvalue of an integer matrix (and therefore an algebraic number). The fixed-point construction allows us to get any computable number. Here is the formal statement: for any computable real $\alpha \in [0, 1]$ there exists a tile set $\tau$ divided into A- and B-tiles such that for any $\varepsilon > 0$ there exists $N$ such that for all $n > N$ the fraction of A-tiles in any $\tau$-tiling of $n \times n$-square is between $\alpha - \varepsilon$ and $\alpha + \varepsilon$.

References


Physical Experiments as Oracles

Edwin Beggs ∗†  José Félix Costa †‡  John V. Tucker †

1 Introduction

Over the past six years we have developed a methodology, some mathematical theory, and several applications, exploring physical foundations for computability and complexity. Enough of our theory and case studies have been published, sometimes in media remote from theoretical computer science, to warrant a short review of our approach and results1. Our aim of this review is to arouse curiosity, not to satisfy it: we will refer to our published papers for expositions, technical details and references to related work. Although we have read widely, we will be delighted to receive information about research related to our programme. Here we will concentrate on the theory of using a physical experiment as an oracle to an algorithm.

∗Corresponding author.
†School of Physical Sciences, Swansea University, Singleton Park, Swansea, SA3 2HN, Wales, United Kingdom. Email: E.J.Beggs@Swansea.ac.uk and J.V.Tucker@swansea.ac.uk.
‡Department of Mathematics, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisboa, Portugal, and Centro de Matemática e Aplicações Fundamentais do Complexo Interdisciplinar, Universidade de Lisboa, Lisbon, Portugal. Email: fgc@math.ist.utl.pt.
1We thank Prof. Rozenberg for the invitation to discuss our work in the Bulletin.
1.1 Our questions

Our research programme began by focussing on two questions:

Consider a physical experiment in which some independent physical quantities $x_1, x_2, \ldots$ may vary and a series of corresponding behaviours $y_1, y_2, \ldots$ can be observed. The experiment defines a function $y_i = f(x_i)$ or, more generally, a relation $r(x_i, y_i)$ on real numbers, for $i = 1, 2, \ldots$.

**Question 1.** How can we use such an experiment to compute a function $f$, or decide a relation $r$, on a set of data? How do we measure its performance?

To answer the question we must think precisely about data representation by physical quantities, experimental equipment and experimental procedures; and we must gain a thorough understanding of one (or more) physical theories. This form of calculation, based directly and exclusively upon a physical system, we called experimental computation. Now, for any particular type of experimental computation we can ask the question:

**Question 2.** Is experimental computation equivalent to algorithmic computation? If not, what is the difference?

Specifically, can we determine what functions and sets a particular class of experiments computes? The question is related to all sorts of physical aspects of computing to be found in research on Natural and Unconventional Computing. In Logic, one recalls Kreisel’s questions and speculations in [21].

The papers that focussed on these physical issues are [12, 13, 14, 15, 16, 17].

Building upon our first steps at making physical theories of computing, we considered combining experimental and algorithmic computation. Our third question is an extension of the last:

**Question 3.** If a physical experiment were to be coupled with algorithms, would new functions and relations become computable or, at least, computable more efficiently?

To answer this question we pursued the idea of using experiments as oracles to algorithms. This has proved to be interesting and productive, both conceptually and technically. Changing the direction of thought, from computation to physics, we have begun to answer this question:

**Question 4.** If a physical experiment were to be completely controlled by an algorithm, what effect would the algorithm have on the physical measurements made possible by the experiment?

Our papers on these hybrid issues are [5, 4, 6, 10, 7, 8, 11, 9].

1.2 Our programme

We introduced the idea of experimental computation to capture a large and diverse set of examples, old and new, and to formulate our general approach to
analysing computation as physical process. Examples abound in modelling natural systems and technologies for designing machines. Experimental computation involves choosing a physical system, which may be a part of nature or a machine, and is therefore dependent on specific physical theories; these are needed to specify and reason about what can be computed. Thus, the experiments are idealised and abstract experiments — “gedankenexperimente”. We developed a number of general principles and case studies to explore and shape some theory. For completeness, we give the principles in Section 2, but there is no substitute for the looking at their origins and use in [12, 13, 14, 15, 16, 17].

As a case study, an experiment was devised to measure the position of the vertex of a wedge to arbitrary accuracy, by scattering particles that obey some laws of elementary Newtonian dynamics [15]. Let SME denote this scatter machine experiment. The SME was put under a theoretical microscope: the Newtonian theory was specified precisely and theorems were proved that showed that the experiment was able to compute numbers measuring positions that were not computable by algorithms. Indeed, the experiment SME could, in principle, measure any real number to any accuracy. Thus, [15] contains a careful answer to Question 2 above, in the negative: the experiment is capable of measuring wedge positions that are non-computable numbers. We argued why this cannot be ruled out by the physical theory.

To pursue Questions 3 and 4, we imagined using a physical experiment as an oracle to a Turing machine, which on being presented with, say, $x_i$ as its $i$-th question, returns $y_i$ to the Turing machine. Now, choosing a physical experiment to use as an oracle is a major undertaking. The experiment belongs to physics and involves concepts from the study of experimental computation (equipment, experimental procedure, measurement, observable behaviour, etc.). The connection between the abstract computing device and the physical experiment is defined by a protocol through which each query communicates information to fix the parameters of an experiment and, afterwards, receives an answer from the oracle to the device (e.g., yes or no).

We began to address Question 3 by using the SME as an oracle to a Turing machine and attempting to classify the computational power of the new type of analogue-digital system. Given the strong results in [15], the SME seemed an interesting choice of oracle that should enhance the computational power and efficiency of the Turing machine. We found that physical oracles demanded changes to some basic assumptions. For example:

(a) the protocol consumed resources, especially time, that had to be integrated into the complexity measures;
(b) the setting of initial conditions by queries could be exact or error prone;
(c) the experiments could have non-deterministic behaviour.

In [4, 7], we began the study of physical oracles by formulating certain prop-
erties of protocols and precision, and by determining the computational power of these machines in terms of non-uniform complexity classes (see [3]).

In [8] we studied in depth a second Newtonian experiment to measure inertial mass, called the collider machine experiment, CME. The protocol necessary for the CME proved to be particularly revealing physically. In particular, we realized that we could view a Turing machine with an oracle in a rather different way. Rather than supposing the oracle was there to boost the computational power of the Turing machine, we assumed the Turing machine could be regarded as an idealised experimenter performing experiments in Nature\(^2\).

Thus, we have two questions and two reasons for interesting ourselves in physical oracles for algorithms: they help us think about the computing power of physical technologies and the laws of physical measurement.

2 Computation and physical systems

We summarise six methodological principles for the theoretical investigation of Questions 1-4.

2.1 Methodological principles for experimental computation

The idea of experimental computation is to attempt to analyse physical models of computation independently of the theory of algorithms. Physical theories play a fundamental role in understanding experimental computation, which we have discussed at length elsewhere [13, 14]. To seek conceptual clarity, and mathematical precision and detail, we proposed, in [13, 14], the following four principles and stages for an investigation of any class of experimental computations:

**Principle 1. Defining a physical subtheory:** Define precisely a subtheory \( T \) of a physical theory and examine experimental computation by the systems that are valid models of the subtheory \( T \).

**Principle 2. Classifying computers in a physical theory:** Find systems that are models of \( T \) that can through experimental computation implement specific algorithms, calculators, computers, universal computers and hyper-computers.

**Principle 3. Mapping the border between computer and hyper-computer in physical theory:** Analyse what properties of the subtheory \( T \) are the source of computable and non-computable behaviour and seek necessary and sufficient

\(^2\)This is not unlike the role of algorithms in learning theory (e.g., [19]).
Principle 4. Reviewing and refining the physical theory: Determine the physical relevance of the systems of interest by reviewing the truth or valid scope of the subtheory. Criticism of the system might require strengthening the subtheory $T$ in different ways, leading to a portfolio of theories and examples.

To study experimental computation and seek answers to Questions 1 and 2, the key idea is to lay bare all the concepts and technicalities to be found in examples by putting them under a mathematical microscope. Our methodology requires a careful formulation of a physical theory $T$, which can best be done by axiomatisations, ultimately formalised in a logical language, i.e., a formal specification of a fragment of the physical theory. We analyse the computational behaviour of classes of systems that obey the laws of $T$ and so study $T$-computability. Our approach has been applied in a new discussion of the physical basis of the Church-Turing Thesis in Ziegler [23]. A language for Newtonian experimental procedures was described in [16].

2.2 Methodological principles for combining experiments and algorithms

Next, we extend our methodology to consider the interaction between experiments and algorithms, and answer Questions 3 and 4.

First, consider using an experiment as a component to boost the performance of an algorithm or class of algorithms. In this case, computations involve some form of protocol for exchanging data between physical system and algorithm. A simple general way to do this is to choose an algorithmic model and incorporate the experiment as an oracle. There are many algorithmic models but the advantage of choosing Turing machines is their rich theory of computational complexity.

Suppose we wish to study the complexity of computations by Turing machines with experimental oracles. Given an input string $w$ over the alphabet of the Turing machine, in the course of a finite computation, the machine will generate and receive a finite sequence of oracle queries and answers. Specifically, as the $i$-th question to the oracle, the machine generates a string that is converted into a rational number $x_i$ and used to set an input parameter $p_i$ to the equipment. The experiment is performed and, after some delay, returns as output a rational number measurement, or qualitative observation, $y_i$, which is converted into a string or state for the Turing machine to process. The Turing machine may pause while waiting for the oracle. In summary:
Principle 5. Combining experiments and algorithms:  Use a physical system as an oracle in a model of algorithmic computation, such as Turing machines. Determine whether the subtheory $T$, the experimental computation, and the protocol extends the power and efficiency of the algorithmic model.

Secondly, consider the nature of an experimental procedure for making a measurement. The process of calculating initial conditions and performing an experiment step by step and interpreting the results can be expressed as a sequence of commands and rules. We imagine a human experimenter following an experimental procedure made up of basic operations defined by a physical theory $T$. The analysis can be compared with Turing’s analysis of the human computer, which resulted in the Turing machine. If the $T$-commands are coded on the tape then the Turing machine represents an abstract model of the procedure. Of course, today many experiments are fully computer controlled.

Principle 6. Algorithms controlling experiments:  Use a model of algorithmic computation, such as Turing machines, to control a physical system. Determine whether the subtheory $T$, the experimental procedure and equipment, and the protocol extends or limits the accuracy and efficiency of the physical experiment to make measurements.

In [8, 10] we introduce this principle and the Question 4.

Our experiments are ideal. However, the ontology of our gedanken experiment means that there is no fundamental difference between the ideal experiments defined by physical theory and the Turing machine. Like the Turing machine, they unify the essential features of examples, and map the limit of physical reality.

3 The view from computation theory

Computational complexity theorists have been using, unconsciously, some forms of gedanken experiment, namely, throwing a fair coin. Using Turing machines and diverse accept/reject criteria, they measure computational power, say in polynomial time, by defining complexity classes such as $P$, $NP$, $PP$, $BPP$, $R$, $ZPP$, etc.

To introduce what physical oracles involve technically, let us dismantle the standard black box idea of an oracle.

Consider the classical model of a Turing machine with an oracle. We imagine the oracle to be an unknown external device and we model it as a set $A$. The machine queries the oracle $A$ with a string $w \in \Sigma^*$, and the oracle answers “Is
The Bulletin of the EATCS

In one computational step. Now, consider a particular type of oracle, the tally oracle. A tally oracle is a subset of $\Sigma^*$, with $\Sigma$ a single letter alphabet, e.g., $\Sigma = \{0\}$. The class of sets decidable in polynomial time by Turing machines consulting tally oracles is the well known class $P/poly$; see [3] and Section 4.

**Oracle as a real number.** Let us interpret the tally set $A$ as a real number $x$ in the unit interval $[0, 1]$ in its binary expansion: for any positive integer $i$, if $0^i \in A$, then the $i$-bit of the binary expansion of $x$ is 1, otherwise it is 0. Suppose the tally set oracle $A$ is replaced a real number $x$.

**Oracle as a physical quantity.** The number $x$ can be seen as a value of a physical quantity (ignoring units of physical quantities) such as a distance between two points, an electric charge in an electrostatic field, or a mass of a particle. For definiteness, suppose $(x \equiv) \mu$ is the measure of a mass defined by (a fragment of) Newtonian dynamics $T$. Suppose the abstract oracle $A$ is replaced with an unknown physical experiment of measuring a mass defined by $T$.

**Oracle as an experiment measuring mass.** In $T$ the measure of a mass is a collateral effect of combining Newton’s Second and Third Laws of dynamics. We derive the theorem of conservation of linear momentum in $T$ and are able to calculate the value of the mass by an experiment using a proof particle of known mass to collide with the unknown mass. This experiment has centuries of commentaries and discussions, starting with Galileo. But now, as computer scientists hunting for the binary digits of $\mu$, and not simply for a canonical measurement of the kind $\mu \pm \Delta \mu$, an experimental procedure must be found to discover the binary bits of $\mu$ and use them as oracle to the Turing machine. Suppose the oracle is such an experiment.

**Cost of performing the experiment.** We choose a specific experiment for mass such as the collider machine $CME$ in [8]. To measure the real number $\mu$ we have to proceed, step by step, with finite approximations. Thus, besides the unknown mass $\mu$, we use proof particles of dyadic rational masses (easily denoted by finite binary strings). The idea is this: if we project a particle of known mass towards a particle of unknown mass, then the first will be reflected if its mass is less than the unknown mass, and it will be projected forward, together with the particle of unknown mass, if its mass is greater than the unknown mass. There is a bisection procedure which determines, bit by bit, the value of the unknown mass. But here we find another novelty: if we want to read the bits of $\mu$ using such a method then
the time needed for a single experiment is

\[ \Delta t = \left| \frac{1}{m - \mu} \right| , \]

where \( m \) is the mass of the proof particle in that single experiment. The time needed for a single experiment to read the bit \( i \) of the mass \( \mu \), using the proof particle of mass \( m \) of size \( i \) (number of its bits) is at best exponential in \( i \).

This experiment \( CME \) (fully described in [8]) tells us that the time needed to consult the oracle is no longer a single step but \textit{depends on the size of the query}. The Turing machine with the experiment as oracle must also be provided with a \textit{protocol}, i.e., a process that manages the interface and counts the time needed in each consultation of the oracle.

Precision in performing the experiment. In the physical world it is not reasonable that a proof particle of mass \( m \) can be set with infinite precision. Does our measurement of \( \mu \), bit by bit, need us to possess equipment with infinite precision? The answer is \textit{No}. Suppose precision is not infinite but finite and unbounded, i.e., we can be as precise as we need. We have shown that we can continue reading the bits of \( \mu \). The passage from infinite to finite unbounded precision, in setting the mass of the proof particle, is no obstacle to the reading of bits: the same complexity classes are defined and the same protocols can be used to communicate between the Turing machine and the physical experiment. However, if we reject unbounded precision in favour of the more realistic \textit{fixed precision} criterion then we have shown that, using stochastic methods, we are still able to read the bits of \( \mu \). The lack of precision in measurement is not an obstacle to the reading of the bits of \( \mu \).

4 Reflections on computation

Let us consider Principle 5 in 2.2. Our discussion in Section 3 introduced a new complexity theory of Turing machines where oracles (a) need time to be consulted and (b) operate with various forms of precision.

Managing oracles. A protocol manages the interaction with the experimental oracle. First, we have to classify the protocols.

The Turing machine will generate a (dyadic) rational number \( x_i \), which can be used to set an experimental parameter \( p_i \) of \( SME \) in one of two ways: we call the machine \textit{error-free} if \( p_i = x_i \); and \textit{error-prone} if we can ensure that \( p_i \in [x_i - \varepsilon, x_i + \varepsilon] \) for an error margin \( \varepsilon > 0 \). In the error-prone case, we further
differentiate between fixed accuracy, where $\varepsilon > 0$ is fixed for the particular SME, and arbitrary accuracy, where $\varepsilon > 0$ can be made arbitrarily small.

The protocol consumes and measures the time (and, possibly, other physical resources) needed to settle the parameters of the experimental equipment and to perform the experiment. The time is dependent on the size of the Turing machine’s query, which denotes the values of the parameters.

These properties introduce these different types of protocols: polynomial time error-free, exponential time error-free and polynomial time error-prone with arbitrary or fixed precision. The SME possess a natural polynomial protocol whilst that of the CME is exponential.

Finally, we note that with the SME a particle hitting the vertex of the wedge may scatter randomly. Thus, the Turing machines are divided further by their deterministic and non-deterministic oracles.

**Classification.** Taking some inspiration from the technical work of Siegelmann and Sontag [22], we use non-uniform complexity classes of the form $B/F$ and $B//F$, where $B$ is the class of computations and $F$ is the advice class. Examples of interest for $B$ are $P$ and $BPP$; examples for $F$ are poly and $\log^*$. The power of the machines will correspond to different choices of $B$ and $F$. A first result for the SME was this:

**Theorem.** The class of sets which are decidable in polynomial time by error-free deterministic analogue-digital scatter machines with a polynomial protocol is exactly $P/poly$.

For the error-prone machines we proved lower bounds in [4] and, later, upper bounds in [7], which enabled us to complete the classification of the power of the three kinds of scatter machines. For example, for the error-prone machines, we prove:

**Theorem.** The class of sets which are decidable in polynomial time by error-prone arbitrary precision deterministic analogue-digital scatter machines with a polynomial protocol is exactly $P/poly$.

Thus, there is no difference in computational power between using exact queries and approximate queries having arbitrary precision. This is not the case for fixed finite precision, which computes less for the scatter experiment SME.

The computational power reduction of reading capabilities of physical equipment through the two experiments we analysed so far in [5, 4, 6, 10] is summarized in the following table.
5 Reflections on physics

Let us consider Principle 6 in 2.2. The idea of using algorithms to completely control physical experiments seems to us to be both simple and radical, and the task of making a theory to explore its meaning and consequences an intriguing challenge. For example, one aim would be:

To create a theory about the nature of physical measurements.

Certainly, the experience of thinking about experimental computation is invaluable; we encounter problems of specifying physical theories, analysing and modelling experimental procedures, equipment, observers and experimenters. The subject is a potent mixture of the philosophical and technical, offering work for philosophers, logicians, computer scientists and physicists. Theoretical studies of measurement seem to be rather diverse.

Let us consider the fascinating paper [18], by the physicists Geroch and Hartle, published in 1986 to celebrate the 75th birthday of John Archibald Wheeler. We will outline some of their main ideas and speculations, which we will reinterpret and formalise using our theory.

Geroch and Hartle start by considering the concept of measurable number in contrast to the concept of computable number:

We propose, in parallel with the notion of a computable number in mathematics, that of a measurable number in a physical theory. The question of whether there exists an algorithm for implementing a theory may then be formulated more precisely as the question of whether the measurable numbers of the theory are computable.

Then they add some considerations on numbers being measurable and/or computable:

We argue that the measurable numbers are in fact computable in the familiar theories of physics, but there is no reason why this need be the case in order that a theory have predictive power. Indeed, in some recent formulations of quantum gravity as a sum over histories, there are candidates for numbers that are measurable but not computable.

They introduce the notion of observer and physicist for the purpose of measuring physical variables:
Regard number \( w \) as measurable if there exists a finite set of instructions for performing an experiment such that a technician, given an abundance of unprepared raw materials and an allowed error \( \varepsilon \), is able by following those instructions to perform the experiment, yielding ultimately a rational number within \( \varepsilon \) of \( w \).

The set of instructions which Geroch and Hartle refers to, together with some memory to take account of intermediate calculations, we can be interpret as a Turing machine. The Turing machine represents the physicist, or the observer, or the experimenter. Thus, we propose the assumption:

**Postulate 1** The physicist is modelled by a Turing machine. The measuring process is controlled by an algorithm that runs on the machine, generating the atomic instructions to be performed at each step of the experimental procedure.

This postulate says that the experimenter cannot escape the logic of following rules as formalised by computability theory. The logic of experimental procedures can be captured by a Turing machine.

A point not considered in [18] is that not all measurements are possible. Assuming the physicist to be a Turing machine, then the limits of computation of the Turing machine imply limits on measurements; and, therefore, on the nature of physical experiments.

The accuracy \( \varepsilon \) is to be understood as arbitrarily small. As we will see, our work makes the concept of measurable as precise as the concept of computable, which was not the intention of [18]:

"Measurable" is analogous to, although of course much less precise than, "computable". The technician is analogous to the computer, the instructions to the computer program, the "abundance of unprepared raw materials" to the infinite number of memory locations, initially blank. Indeed, one can think of the measurable numbers as those that are "computable" using an analog, rather than digital, computer.

Geroch and Hartle stress need for a theory to specify a *gedanken experiment* as follows:

The notion "measurable" involves a mix of natural phenomena and the theory by which we describe those phenomena. Imagine that one had access to experiments in the physical world, but lacked any physical theory whatsoever. Then no number \( w \) could be shown to be measurable, for, to demonstrate experimentally that a given instruction set shows \( w \) measurable would require repeating the experiment an infinite number of times, for a succession of \( \varepsilon \)s approaching zero. One could not even demonstrate that a given instruction set shows measurability of any number at all, for it could turn out that, as \( \varepsilon \) is made smaller, the resulting sequence of experimentally determined rationals simply fails to converge. It is only a theory that can guarantee otherwise.

Now, how does the Turing machine communicate with Nature? We believe that this interaction is captured by the concept of the continuing evolution of a
Postulate 2 The measurement process is taken to be an oracle to a Turing machine. The interaction is achieved through a protocol. After each consultation, the oracle may provide one bit of the measurement. This bit is also information that provides the necessary information to the machine to proceed with the desired computation and experimental procedure.

These considerations are enough for them to prove theorems such as: Every computable number is measurable. Then the authors ask the following question:

We now ask whether, conversely, every measurable number is computable — or, in more detail, whether current physical theories are such that their measurable numbers are computable. This question must be asked with care.

Actually, the question received a very careful answer in our [15]: the experiment SME demonstrates that there numbers that are measurable in Newtonian dynamics but that are not computable.

Much more can be said in answer to this question. According to a our framework all depends upon the physical theory chosen. For Newtonian mechanics we have shown that for some experiments quantities are always measurable (see [15]) whilst for others there are quantities that are not always measurable (see [10, 8]). Our technical results can be used to show that the task of measuring quantities in physics, let us say in polynomial time, can be classified by well known complexity classes.

Principle 6 and the postulates leads to a deeper understanding of experimenters and experiments which impose a theoretical and absolute limit on the measurability of a physical quantity. Indeed, we are exploring the conjecture that Any quantity in physics may not be always measurable.

The ideas of protocol and precision discussed earlier play a role in the model. In [20, 1], inter alia, we find early explicit statements on measurability, accuracy, and time complexity of physical quantities. For instance, Bachelard starts digressing about this fact in his Philosophy of No ([1]) and, in [2], he states that improving accuracy of an experiment increases exponentially the time needed.

In the oracle model, we are not interested in the time taken to build the experimental apparatus from the unprepared raw materials a la Geroch and Hartle, nor in the time taken to tune the instruments to some precision. We are interested in the run time of the physical experiment.

6 Conclusion

Ours is one among a number of research programmes involving physical systems and computability. Our objectives are foundational rather than technological.
The Bulletin of the EATCS

In [6, 11] we argue that algorithms with physical oracles occur naturally in Physics. In [9] we have studied relativisations of the $P = NP$ problem to physical oracles.

We have considered several standard experiments in Physics measuring distance, inertial mass, resistance, temperature, the ratio $e/m$ of an elementary particle in the classic or quantum Coulombian field, and the Brewster angle in optics. In all these experiments we found that physical experiments have an exponential time protocol. Our results are intended to be a theoretical best case, where introducing more realism would only serve to make the result worse. The project of finding physical systems which allow us to measure some physical quantity more accurately and efficiently — e.g., allowing us to halve the error without doubling the time taken (or multiplying by some other fixed factor) — we conjecture to be condemned to failure.

Our work with Principle 6 leads us believe that measurability in Physics is subject to laws which are co-lateral effects of the limits of computability and computational complexity. Our model imposes limitations on the physics we used to describe it. Not all masses can be known, not because of the limitations in measurements due to experimental errors, but because of essentially internal logical limitations of the theory. The mathematics of computation theory does not allow the reading of bits of physical quantities beyond a certain limit. Quantities cannot be measured with infinite precision, not because of the limitations of the physical apparatus but, more deeply, because essentially computational reasons. These unmeasurables allow the definition of quanta of energy for the classical physical world, accompanied by uncertainty principles, courtesy of the theory of computation.

Edwin Beggs, José Félix Costa and John Tucker would like to thank EPSRC for their support under grant EP/C525361/1.

References


The Bulletin of the EATCS


THE PUZZLE CORNER

BY

LAURENT ROSAZ

LRI, Orsay CNRS-Université de Paris Sud
Bât 490, 91405 Orsay France
Laurent.Rosaz@lri.fr

Readers are invited to send comments, and to send exercises, even if they don’t know the answer. Write to Laurent.Rosaz@lri.fr.

84 Monochrome-rectangle-free Black and White grids

The squares of an $n \times n$ grid $G$ are colored in either white or black. A monochrome rectangle is given by four indices $l_1 < l_2$, $c_1 < c_2$ such that the four squares $(l_i, c_j)$ are the same color.

For example, the left grid has two white rectangles (lines 1 and 2, columns 3 and 4, and lines 2 and 4, columns 1 and 3) and two black ones. The other two grids are monochrome rectangle free.

Is there a $5 \times 5$ monochrome-rectangle-free black and white grid?
Solutions to Previous Puzzles

83 The penguins

Penguins are wandering on the ice. Penguins have colors. There are $N$ different colors. From time to time, two penguins meet, and we assume that every two penguins will eventually meet infinitely often. When two penguins meet, their colors may change: if their colors are different, they don’t change. If they are the same, then the colors change to new colors which depend on their common previous color (for example, two red penguins will become a red one and a green one, two green penguins will become two blue ones, etc.).

Right now, there are $N$ penguins, one of each color. So the colors cannot change.

You want to introduce an $(N + 1)^{th}$ penguin of color $c$. Then you will wait till this penguin meets the other $c$-colored penguin. From then, you wait till there is again a $c$-colored penguin (if this ever happens) and you withdraw one $c$-colored penguin. Then you wait till there is again one penguin of each color (if this ever happens).

Show that there is a color $c$ such that you can introduce a $c$-colored penguin, so that you will indeed end up with one penguin of each color.

Solution:

Let us represent the number of penguins of each color in a configuration $C$ by the vector $V_C = (P_1, ..., P_N)$ where $P_i$ is the number of penguins of color $i$.

Let us denote by $1_i$, the vector which is 1 at coordinate $i$ and 0 elsewhere. For each color $c$, let $W_c = 1_{c_1} + 1_{c_2} - 2 * 1_c$ where $c_1$ and $c_2$ are the colors of two penguins which were of color $c$ and which met. (This vector may contain one -2 and two 1, or one -1 and one 1, or may be the null vector, depending on whether $c_1, c_2$ are $c$)

When you are in configuration $C$, the next event can be the meeting of two penguins of color $c$ if $(V_C)_c \geq 2$, and the new configuration is represented by $V'_C = V_C + W_c$.

Note that $\sum_i (V_C)_i$ is constant during an execution, it is the number of penguins.

Lemma There is a non-null linear combination of the $W_c$ with non-negative integer coefficients, which is null.

(Note: These $N$ vectors are in the space $F : \{(x_i)_{i} | \sum_i x_i = 0\}$, so there are dependant since the dimension of $F$ is $N - 1$, thus there is a null non trivial linear combination of them. This doesn’t help because we require the coefficient to be non negative)
Proof of the lemma Consider a vector $W_0$ with non negative integer values, such that $\sum c(W_0)c > N$. Due to the later property, there is a $c$ such that $(W_0)c \geq 2$, pick up such a $c$, and let $V_1 = V_c$ and $W_1 = V_1 + W_0$. Since $\sum c(W_1)c = \sum c(W_0)c > N$, you can proceed similarly to build $V_2$ and $W_2$, and next, by induction $(V_n)c \in [1,\infty]$ and $(W_n)c \in [0,\infty]$.

Let $Z$ be $\{v|v$ is an $N$ vector with non negative integer coordinates, and $\sum c(v) = \sum c(W_0)c\}$. The $W_n$’s are in $Z$, and $Z$ is finite, thus there are $i < j$ such that $W_i = W_j$. The requires linear combination is $\sum_{i=1}^{j} V_k$.

The lemma is proved

Let $F$ be a minimal (for inclusion) non empty multiset of colors such that $\sum c(F)c = 0$ (the existence of $F$ is guaranteed by the lemma).

Right now, there are $N$ penguins, one of each color, the configuration is $\sum c I_c$. Let $c_1$ be a color in $F$. Add an $(N + 1)^{th}$ $c_1$-colored penguin, the configuration is now $K_0 = I_{c_1} + \sum c I_c$.

Wait till two penguins with the same color meet, it will be the penguins with color $c_1$. The configuration will be now $K_1 = K_0 + V_{c_1}$.

If a $c_1$ colored penguin appears, hold it back. We describe what happens next by pseudo-code:

$i \leftarrow 1$
$K_i \leftarrow K_0 + V_{c_1}$
$F_i \leftarrow F - \{c_1\}$
$G_i \leftarrow \{c_1\}$

invariants : $(F_i, G_i)$ is a partition of $F$, and $K_i = K_0 + \sum c G_i V_c$

While the configuration $K_i$ is not equal to $K_0$ again, do:

Wait till two penguins with the same color meet (there are such penguins since $K_i \neq K_0$). Let $c_{i+1}$ be the color of these penguins. Since there were two penguins which met, it means that $(c_{i+1} \neq c_1$ and $(K_i)c_{i+1} \geq 2)$ or $(c_{i+1} = c_1$ and $(K_i)c_{i+1} \geq 3$ (recall that you hold one $c_1$-colored penguin if there are any). Thus $(K_i)c_{i+1} > (K_0)c_{i+1}$, that is $(\sum c G_i V_c)c_{i+1} > 0$, thus $(\sum c F_i V_c)c_{i+1} < 0$ (recall that $F_i = F_i + G_i$ and that $\sum c F_i V_c = 0$), and thus there is a $c$ in $F_i$ such that the $(c_{i+1})^{th}$ coordinate of $V_c$ is negative. That $c$ can be no color but $c_{i+1}$. We have proved that $c_{i+1} \in F_i$.

\{
$c_{i+1} \leftarrow$ the color of the two penguins, as described above
$K_{i+1} \leftarrow K_i + V_{c_{i+1}}$
$F_{i+1} \leftarrow F_i - \{c_{i+1}\}$
$G_{i+1} \leftarrow G_i + \{c_{i+1}\}$
$i \leftarrow i + 1$
\}
Since $F_i$ is decreasing, the loop must end. When it ends, you have reached the goal.

Note 1: note that the color of the penguin to inject can be any color in many cases. Draw the graph whose vertices are the color, and an arrow goes from $c$ to $c'$ if $c'$ is the color of at least one penguin after two penguins of color $c$ have met. The color of the injected penguin can be any color if that graph is strongly connected. In general, it can be any color in strongly connected components which are "at the bottom" (i.e., no arrow goes from inside this SCC to outside this SCC).

Note 2: Following the proof, the number of monochrome penguin meetings before you get back to the initial configuration is less than $\text{card}(Z)$ which is exponential in $N$. Note that this number may be indeed exponential, consider for example the rule to be that when two $i$-colored penguins meet, one of them gets 0-colored, the other one gets $((i+1) \mod N)$-colored.
E VOLUTION OF M U TATING S W A R D W A R E*

Gregory Chaitin†

Abstract

We propose using random walks in software space as abstract formal models of biological evolution. The goal is to shed light on biological creativity using toy models of evolution that are simple enough to prove theorems about them. We consider two models: a single mutating piece of software, and a population of mutating software. The fitness function is taken from a well-known problem in computability theory that requires an unlimited amount of creativity, the Busy Beaver problem.

Key words: evolution, random walk, software space, Busy Beaver function

1 Introduction

This paper proposes modeling biological evolution as mutating software. However at the level of abstraction of this paper, biological evolution and mathematical creativity are not so different.

I have been interested in theoretical biology for a long time [1]. However I could not find a way to go forward. Here we propose an approach that I feel shows promise, but much remains to be done. We only present some preliminary results.

The immediate stimulus for this paper was Berlinski’s entertaining polemic [2], in which he presents a number of criticisms of Darwinian evolution, a number of perplexing aspects of evolution that he feels are not yet well explained by Darwin’s theory.

Here are five areas in which I believe that thinking of DNA as mutating software can be helpful. (Software mutations may be point mutations, such as changing or adding a single bit, or they may be high-level, such as copying a subroutine and then modifying it.)

* Talk given Friday October 10, 2008 at the IBM Watson Research Center in Yorktown Heights, NY. The author wishes to thank his colleagues Charles Bennett and David DiVincenzo for their helpful comments.
† IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598, U.S.A., chaitin@us.ibm.com.
1. **Why are there so many missing intermediate biological forms?** This is not a problem, because a small change in a program can produce an enormous change in its output. Change in output does not have to be continuous, it can be extremely discontinuous.

2. **How are sudden major steps in evolution possible, for example, the transition from single-celled organisms to multi-cellular organisms?** What is happening here is that the main program (a single cell) becomes a subroutine in a larger multi-cellular organism. Changing the main program into a subroutine that is called several times is not a big software change.

3. **Why is there so much fractal, hierarchical structure in organisms?** This is just an extension of the previous point. It’s recursive, deeply-nested subroutine structure, which is a good way to build a large piece of software out of smaller pieces of software.

4. **Why does complexity increase?** An increase in the size of a software organism may be due to a genuine increase in function but may also be due to software bloat, the familiar fact that it is easier to add new code than to change existing code.

5. **Why do we need DNA? In other words, why is the genotype different from the phenotype?** Well, if we are mutating software written in a very redundant high-level language convenient for humans, the mutation distance between useful organisms will be much larger than if we are mutating a much more concise representation of these algorithms more similar to a binary machine language. So a way of obtaining redundant programs from compressed versions of them will probably evolve, since it greatly increases evolvability by decreasing the mutation distance between useful software.

   On the other hand, if we are mutating software written in the extremely compact kind of representations used in algorithmic information theory [3], then there is no need for DNA and the genotype can remain the same as the phenotype.

   Based on these general considerations, we propose studying the evolution of mutating software. We already made this proposal in [4], but there we could not see where the dynamics comes from. We could not see how to get our software organisms to evolve.

   Now I’ve found a way to do that. In this paper we describe some simple models where you can show that complexity increases due to increased function, not due to software bloat (point (4) above).
2 A problem requiring creativity: Naming big numbers

To get evolution, we must give our software organisms something challenging to do. Here is a simple mathematical problem requiring unlimited creativity, naming big numbers.

For example, \(100^{100}\) is certainly a big number, but we can do much better than that.

Following Hodges [5], write \(n \uparrow m\) instead of \(n^m\). Then define \(n \uparrow \uparrow m\) as follows:

- \(n \uparrow \uparrow 0 = 1\),
- \(n \uparrow \uparrow (m + 1) = n \uparrow (n \uparrow \uparrow m)\).

Similarly, define \(n \uparrow \uparrow \uparrow m\) as follows:

- \(n \uparrow \uparrow \uparrow 0 = 1\),
- \(n \uparrow \uparrow \uparrow (m + 1) = n \uparrow \uparrow (n \uparrow \uparrow \uparrow m)\).

Continue in this fashion with four up arrows, five, etc.

Going beyond Hodges, define \(n \uparrow \cdots \uparrow m\) to be \(n \uparrow \cdots \uparrow n\) with \(m\) up arrows! What is the value of \(999 \uparrow \cdots \uparrow 999\)?

Unlimited creativity is required for this problem, because for any scheme for naming large numbers, one can come up with a better scheme, just like we improved on Hodges.

(See also Steinhaus [6], Knuth [7], Davis [8].)

Naming big numbers is closely related to a classical problem in computability theory, the Busy Beaver problem, and the Busy Beaver function. If you are naming numbers informally as we have just done, then a Busy Beaver function of \(N\) can be loosely defined as the biggest positive integer you can name in \(\leq N\) symbols or characters of text.

If you are using a programming language to name positive integers, then the Busy Beaver function is the greatest positive integer that can be produced by a program of size \(\leq N\) bits (programs that calculate a single integer and then halt). Busy Beaver functions grow faster than any computable function of \(N\), which shows that naming big numbers requires an unlimited amount of creativity.

For more on this, see Aaronson [9] and Wikipedia [10].
3 The formal setting

In a nutshell:

- binary program = genotype = phenotype,
- positive integer output by program = fitness.

3.1 Software space = binary programs for calculating a positive integer

All finite bit strings are valid programs, but some may never produce any output.

We have to pick a universal Turing machine = a general-purpose programming language. I’ll use the universal Turing machine that I’m most familiar with, the one $U$ in [3], which has the property that for any other Turing machine $C$, there is always a prefix $\pi_C$ such that the output $U(\pi_C p)$ produced when the concatenation of $\pi_C$ with $p$ is run on $U$, is the same as the output $C(p)$ produced when $p$ is run on $C$. This shows that the programs for $U$ are concise, or, more precisely, not much bigger than the programs for $C$.

However, we need to modify the universal machine $U$ of [3] so that it runs forever without producing any output or else produces a single positive integer 1, 2, 3, as output. This is easy to do by filtering the output of $U$. In addition, $U$ requires programs to be self-delimiting, and not all bit strings are valid programs. Here we need all finite bit strings to be valid programs. This can be done by modifying $U$ so that it ignores extra program bits and loops forever if it runs out of program bits.

Since $U$ is universal, our software space includes all possible algorithms for calculating a positive integer.

3.2 Point mutations and mutation distance

Point mutations: change a bit, delete a bit, or insert a bit. (Some authors allow adjacent bits to be interchanged, but I’ll omit that here.)

The most straight-forward way of defining a metric on software space is as the number of point mutations required to get from one organism to another.

That’s the general idea, but sometimes we need a more subtle way to define mutation distance, as $-\log_2$ of the probability of getting from one organism to another via a single mutation. That is the right way to think about mutation distance if you can go from any organism to another in one mutation with small but non-zero probability, which we need to do in model 1 to avoid getting stuck on a local fitness maximum.
3.3 The fitness function: Naming big numbers

The bigger the positive integer that is produced by a program, the fitter it is. If it never produces any output because it never halts, then it is totally unfit.

3.4 We need an oracle for the halting problem

In our models we will need to use an oracle for the halting problem, because if a random mutation would give us a program that never produces any output, we want to be able to skip it. If it does produce output, we can run the program and see how fit it is.

4 Model 1: A random walk in software space

In this model we consider a single software organism, initially the empty program = zero-length bit string. At each step \( N \), we pick a mutation at random, and check if the resulting organism is more fit. If so, this organism becomes our new step \((N + 1)\)th organism. Otherwise we pick another mutation at random and continue as before.

The problem here is to get the random walk to cover the entire software space, i.e., be ergodic. If we pick a point mutation at random this will not happen. There is also the problem of being stuck in a local fitness maximum.

To avoid these problems, we can’t just use point mutations, we need to do something more sophisticated. We need a small but non-zero probability of going from any software organism to any other.

One way to get this to work, which just happens to be the first way that I could think of, but which is no doubt only one of many possible ways to accomplish this, is as follows:

- To get each new organism, use a single point mutation with probability \(1/2\), use two point mutations with probability \(1/4\), three with probability \(1/8\), etc.

- Also, bias the point mutations to change the beginning of the program. This is a good strategy since our universal computer \(U\) reads a self-delimiting prefix from the beginning of the program string and then runs it (see [3]).

- The point mutation will delete, flip, or insert a bit at the first bit of the program with probability \(1/2\), it will make a change at the second bit with probability \(1/4\), at the third bit with probability \(1/8\), etc.
If the rules of the game are set up in this way, a single mutation consisting of many point mutations will eventually insert a prefix at the beginning of the software organism that computes an extremely large positive integer and that ignores the rest of the program string. The result is that one can show that with high probability the fitness of our software organism will grow faster than any computable function of the step \(N\), which shows that genuine creativity is occurring.

I omit the detailed calculations and estimates.

## 5 Model 2: Evolution in parallel

In this model we consider a population of software organisms, not a single organism. We start as before with a trivial program, and at each stage add to our population all the software organisms that are one point mutation away from the organisms in our current population. More precisely, each organism gives birth to all those organisms one point mutation away. So by stage \(N\) our population will include all \(N\)-bit software organisms, since we can add a bit at each stage.

Furthermore, fit organisms have many siblings. When an organism is added to the population, we check its fitness. If this is \(K\), we add \(K\) additional copies of that organism to our population.

Programs that produce extremely large numbers \(K\) will quickly predominate. In fact, at stage \(N\) the organism with the most siblings will be the \(\leq N\)-bit program that calculates the biggest number \(K\). This value of \(K\) is by definition the Busy Beaver function of \(N\) bits, the largest output that can be produced by a program \(\leq N\) bits in size, and grows faster than any computable function of \(N\). This value of \(K\) is also the greatest positive integer with program-size complexity (as defined in [3]) \(\leq N\).

Model 2 is simpler than model 1, and it evolves more quickly, because it is evolving in parallel. Note that this is a deterministic model. However, the history leading from the initial organism to each individual organism at stage \(N\), will still look like a random walk.

## 6 Model 3: The trivial model

Let me now criticize our two previous models. Here is another model of evolution. At stage \(N\) look at all programs \(\leq N\) bits in size and select the one that produces the biggest output, which is in fact the Busy Beaver function of \(N\) bits, the largest output that can be produced by a program \(\leq N\) bits in size.

This is even simpler than our previous model, and does just as well. Why did we take the trouble to formulate models 1 and 2, if model 3 is simpler and does
just as well?

The answer is that the problem with model 3 is that it is not at all biological in
spirit. This is not how Nature searches through the space of all possible organisms.

7 Discussion

As the trivial model shows, if you have an oracle for the halting problem (and
all three of our models do), it is easy to obtain concise names for extremely big
numbers. It is the way the search through software space is done that is biological,
not the organism that you come up with. (Although the way the search is done
will influence the kinds of organisms that you are likely to get.)

I should also emphasize that at best our models are the Platonic ideal that
biological evolution attempts to approach, they are not the real thing we see hap-
pening around us. Nature does not have an oracle for the halting problem, and the
number of organisms cannot increase exponentially indefinitely.

Furthermore, different models may be required to shed light on each of the
problems (1) through (5) discussed in Section 1. In this paper, we have only ad-
dressed point (4), why does complexity increase? We get complexity to provably
increase by choosing a fitness function that rewards creativity and by making sure
that our random walks in software space are ergodic, i.e., cover the entire space.

References


pp. 29–30.


REPORTS FROM

CONFERENCES
REPORT ON AFLAS 2008

Workshop Automata, Formal Languages and Algebraic Systems
20–22 September, 2008, Kyōto, Japan

Manfred Kudlek

The DLT 2008 satelliteworkshop Workshop Automata, Formal Languages and Algebraic Systems (AFLAS 2008) took place from September 20-22 at Toritanı 2, Kitashimona-ka-machi, about 40 km Northwest from the centre of Kyōto, still in the town of Kyōto, in a nice landscape. Conference site was the Kyoto Seminar House (KSH). There the participants stayed in traditional Japanese guest houses, in rooms for up to six persons equipped with bast mats on the ground. Also the bath was in traditional Japanese manner, and in the restaurant only Japanese dishes were served. There also are a number of mathematical stone art monuments on the area. In the residence, as customary in Japan, and lecture room we had to change our shoes.

AFLAS 2008 was attended by 38 participants from 12 countries.

\[
\begin{array}{ccccccccc}
\text{CA} & \text{DE} & \text{FI} & \text{HU} & \text{NZ} & \text{RU} \\
1 & 5 & 1 & 5 & 2 & 1 \\
\text{CZ} & \text{ES} & \text{FR} & \text{JP} & \text{PL} & \text{SK} \\
5 & 1 & 2 & 13 & 1 & 1 \\
\end{array}
\]

The scientific program consisted of 6 invited talks and 18 presentations. Details on distribution by countries are given in the following table (C country, I invited).

\[
\begin{array}{ccc|ccc}
\text{C} & \text{I} & \text{C} & \text{I} \\
\hline
\text{AT} & \frac{1}{3} & \text{HU} & \frac{5}{3} \\
\text{CA} & \frac{1}{2} & \text{JP} & \frac{5}{6} \\
\text{CZ} & 3 & \frac{1}{2} & \text{NZ} & 2 \\
\text{DE} & 3 & 1 & \text{RU} & 1 \\
\end{array}
\]

The program can be found at
http://www.cc.kyoto-su.ac.jp/conference/dlt08/

AFLAS was opened on Saturday afternoon by Masami Iro, welcoming all participants. The first invited talk Colonies: Simple Rewriting Agents in a Passive Environment was presented by Alica Klemešová, giving a nice and interesting survey on P systems, P colonies, simulation of regular machines, and colonies of sequential and parallel grammars.
Bakadyr Khoussainov (partly co-authors Greg Hjorth, André Nies, Antonio Montalbán) with Injectivity Problems for Automatic Structures gave a good and interesting second one on various FA’s (Büchi, Rabin), algebraic systems and Borel structures and their relations among each other, finishing with ‘Why is this interesting? Because you are a Mathematician!’.

Manfred Kudlek (co-authors Patrick Totzke, Georg Zetzsche) presented the third one with Concurrent Finite Automata and Related Language Classes, giving an overview on that topic.

On Proper Languages of Lexicalized Automata by Friedrich Otto was an excellent fourth invited talk on finite state acceptors, PDA’s, restarting automata for analyzing natural languages, and on characteristic and proper languages of such automata.

Also the fifth one, Rational Subsets in Finitely Generated Groups, by Markus Lohrey (partly joint with Géraud Sénizergues, Benjamin Steinberg) was a very good presentation on rational and recognizable subsets of monoids, finitely generated groups, their decision problems (membership, submonoid, subgroup), HNN extensions, and amalgamated free products.

Cristian Calude (’co-author Nicholas J. Hay), with Can Peano Arithmetic Prove Randomness?, presented an excellent and interesting sixth invited lecture on randomness, Peano arithmetic and provability in it, existence of non-provable statements for universal prefix-free TM such as randomness for strings, and on computable enumerable random reals being provably random. He finished with ‘Mathematics will be done more and more a hybrid task, human-machine’.

Also to mention are the good presentations by Gerhard Lischke on generalized primitivity and distance of words, by Géza Horváth on new pumping lemmas, by Benedek Nagy on derivation trees for context-sensitive languages, and by Libor Polák (co-author Ondřej Klíma) on relations between languages, algebraic structures and automata.

AFLAS was closed on Monday by Masami Ito, stating that this was his last conference to organize (hopefully not!), and expecting all participants in Japan again, and by Cristian Calude thanking in particular Masami.

In the breaks, coffee, green tea and snacks were offered.

On Sunday evening we had a barbecue on the KSH grounds.

Weather was humid, warm and sunny, with highest temperatures between 25 and 30°C, except for the rainy Sunday.

AFLAS was a successful workshop, well organized, and of high level, in really nice and relaxing surroundings.
REPORT ON CS&P 2008

Intl. Conf. on Concurrency, Specification and Programming

29 September–1 October, 2008, Groß Väter See, Germany

Manfred Kudlek

CS&P 2008, the XVIIth in this series, was held from September 29 - October 1 at Feriendorf Groß Väter See, Germany, about 40 km northwest from the centre of Berlin, a recreation centre where also all participants stayed.

It was organized by INSTITUT FÜR INFORMATIK, HUMBOLDT-UNIVERSITÄT ZU BERLIN. Organizers were HANS-DIETER BURKHARD, LUDWIK CZAJA, GABRIELA LINDEMANN-VON TRZEBIATOWSKI, WOJCIECH PENCZIK, ANDRZEJ SALWICKI, ANDRZEJ SKOWRON, HOLGER SCHLINGloff, and ZBYGNIEW SURAJ.

The conference was attended by 60 participants from 9 countries. The scientific program consisted of 6 plenary lectures and 47 contributions from 12 countries, held in two plenary sessions and 2 parallel tracks, A (Petri nets, models with time, logic, classification, model checking and grammars, and automata) and B (artificial intelligence, applications in medical domains, robotics, software engineering multi-agent systems and applications, objects and cdc2, and knowledge management). Details on statistical distribution of participants and contributions by countries, as well as number of authors, is given in the tables below. The conference was supported by DAAD mutual exchange program between Germany and Poland. The program can be found at

http://www2.informatik.hu-berlin/ki/CSP2008/

CS&P 2008 was opened on Monday by GABRIELA LINDEMANN, welcoming all participants.

The first plenary lecture Synthesis and Analysis of Net Structures and Transition Graphs was presented by LUDWIK CZAJA, MANFRED KUDLEK on the relation between nets and transition systems.
HANS LANGMAACK (co-authors ANDRZEJ SALWICKI, MAREK WARPECHOWSKI) with Some Methodological Remarks Inspired by the Paper "On Inner Classes" by Igarashi and Pierce gave an interesting second one on an inference system for the study of semantics of programming languages.

ZBIGNIEW SURAJ (co-authors MIKHAIL MOSHkov, ANDRZEJ SKOWRON) with On Minimal Inhibitory Rules for almost all k-valued Information Systems gave a third interesting plenary talk on the number of inhibitory rules for construction of classifiers in information systems.

HEINRICH MEllMANN (co-authors DANIEL GOHRING, HANS-DiETER BURKHARD) presented a nice the fourth one, Constraint Based Multiple Target Tracking, on applications for RoBoCup.

MIROSŁAW KURKOWSKI (co-author WOJCIECH PENCZEK) with Verifying Timed Security Protocols via Translation to Timed Automata gave a nice and interesting fifth plenary talk on verification of protocols using product (timed) automata, performed with VeriCS.

The sixth one Experimenting, Observing, Proving - an Old Idea and a Case Study in Program Verification was given by ANDRZEJ SALWICKI (co-authors Grazyna Mirkowska, Oskar Świd). It was a good presentation on the method experiment-observation-hypothesis-proving illustrated for a case study.

Also to mention are the good and interesting presentations by BERNDT FARWER on a theory of task suspension in Petri nets and agent systems, by KAMILA BARYLSKA on persistency in P/T systems, and by IRINA VIBRITSKAITė on various timed event structures.

The conference was closed on Wednesday by HOLGER SCHLINGLOFF and LUDWIK CZAJA, thanking the organizers and participants, in particular GABRIELA LINDEMANN getting a bunch of flowers.

The proceedings, edited by HANS-DIETER BURKHARD, LUDWIK CZAJA, GABRIELA LINDEMANN-VON TKZEBIATOWSKI, WOJCIECH PENCZEK, ANDRZEJ SALWICKI, ANDRZEJ SKOWRON, HOLGER SCHLINGLOFF, and ZBIGNIEW SURAJ, containing all contributions, have been published as Informatik-Bericht Nr. 225, Volumes I, II, III of Humboldt-Universität zu Berlin and are also available at the web site cited above.

In the breaks coffee, tea, soft drinks and snacks were offered. Weather was rather chilly with temperatures below 15° and frequent showers. Breakfast, lunch and dinner were served in the restaurant of the recreation centre. Because of the bad weather the planned barbecue dinner Tuesday had to be shifted into the restaurant. Nevertheless, most of the participants attended a treasure rallye in the woods before dinner.

CS&P 2008 was successful again. The next one will take place somewhere in Poland.
REPORT ON DLT 2008

12th Development of Language Theory
15–19 September, 2008, Kyōto, Japan

Manfred Kudlek

DLT 2008, the 12th in this series of international conferences on theoretical computer science, took place in Kyōto, Japan for the second time, from September 15-19, 2008. Conference site was Kyōto Sangyō University (KSU) in the northern part of the town.

The organizing committee consisted of Masami Ito (co-chair), Yoshiyuki Kunoichi, Peter Leupold, Jianqin Liu, Iván Szabolcs, Masafumi Toyama, (co-chair), Kayoko Tsui, and Soichiro Aogaki, Takafrumi Kobayashi, Hironori Takano.

DLT 2008 was under the auspices of EATCS and sponsored by Kyōto Sangyō University, the Japanese Society for the Promotion of Science, and Kagamur Fondaion of International Science Advancement.

The conference was attended by 84 participants from 18 countries, mainly from Japan and Germany.

The scientific program consisted of 6 invited lectures and 36 contributions selected from 102 submitted papers from 31 countries. Details on distribution by countries and number of authors are given in the two tables below (C country, I invited, S submitted, A accepted).

<table>
<thead>
<tr>
<th>C</th>
<th>I</th>
<th>S</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>3</td>
<td>DE 17</td>
<td>FR 7</td>
</tr>
<tr>
<td>CH</td>
<td>2</td>
<td>ES 3</td>
<td>HU 5</td>
</tr>
<tr>
<td>CZ</td>
<td>7</td>
<td>FI 4</td>
<td>IT 5</td>
</tr>
</tbody>
</table>

The conference was attended by 84 participants from 18 countries, mainly from Japan and Germany.

<table>
<thead>
<tr>
<th>C</th>
<th>I</th>
<th>S</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR</td>
<td>1</td>
<td>IT 1</td>
<td>JP 1</td>
</tr>
<tr>
<td>CA</td>
<td>5</td>
<td>3</td>
<td>JP 1</td>
</tr>
<tr>
<td>HK</td>
<td>3</td>
<td>IT 1</td>
<td>JP 1</td>
</tr>
<tr>
<td>CH</td>
<td>1 7</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>CZ</td>
<td>1 7</td>
<td>2 7</td>
<td>1</td>
</tr>
<tr>
<td>DE</td>
<td>17 7</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>ES</td>
<td>2 6</td>
<td>1 7</td>
<td>1</td>
</tr>
<tr>
<td>FI</td>
<td>1 2</td>
<td>6</td>
<td>NZ 1 1</td>
</tr>
<tr>
<td>FR</td>
<td>12 7</td>
<td>PH 1</td>
<td>UA 1</td>
</tr>
<tr>
<td>HK</td>
<td>1 3</td>
<td>PL 1</td>
<td>1 4</td>
</tr>
<tr>
<td>HU</td>
<td>1 6 3</td>
<td>RO 1</td>
<td>US 4 1</td>
</tr>
<tr>
<td>IN</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bulletin of the EATCS no 97, pp. 171–174, February 2009
© European Association for Theoretical Computer Science
DLT 2008 was opened on Tuesday morning by Masami Ito, talking on DLT, thanking all program and organizing committee members, and informing us that Grzegorz Rozenberg and Oscar Ibarra could not come for other obligations. Unfortunately, he also gave us the sad message that Stefano Varricchio, being an invited speaker, had passed away on August 20, 2008.

Later Toyo Sakai, the president of KSU, gave an address, welcoming the participants, and talking on the history of KSU, its founder Toshima Araki, an astronomer, two of his students being Nobel price winners, and invited scientists as Arnold Tynbree and Hermann Kahn. Just recently Toshhide Masukawa from KSU got the Nobel prize in Physics.

The first invited talk by Juraj Hromkovic (co-author Georg Schnitger) on On the Hardness of Determining Small NFA’s and of Proving Lower Bounds on Their Sizes was an excellent, clear, vivid and interesting lecture, beginning with ‘Let’s start with an experiment’, his personal history of talks, the first at MFCS 1981, 2005 first for public (telling stories ‘from joy to magic’). He also frequently asked the audience like ‘Now I’m requiring an answer, Galina, please’ (number of different rows) or ‘In this case, Mika knows’ (number of submatrices), pointing to him. It was a brilliant survey on minimization complexity of NFA’s and new general mathematical methods such as transformation into special Boolean matrices.

Petr Jančar with Selected Ideas Used for Decidability and Undecidability of Bisimilarity presented a good and interesting second one, starting with ‘Why I wrote this paper’, and talking on decidability of language equivalence for DPDa’s, undecidability of bisimilarity for extensions of PD systems, normal forms, and transformations to other structures.

The third invited lecture Various Aspects of Finite Quantum Automata, by Mika Hirvensalo was an excellent and clear presentation on FQA’s. He started with ‘Why QC? Nondeterministic computing. Thank you!’, switching to end, but then giving some answers as fast algorithms, secure communication, deeper understanding of limits of computation, first realizations like FQA, extensive mathematical machinery, and connection to other topics. He then switched to FQA’s, presenting various models as MO-FQA, MM-FQA, Latvian FQA, and their properties.

Another excellent presentation was the fourth invited lecture The Frobenius Problem and its Generalizations given by Jeffrey Shallit. He started with ‘Talk is sponsored by my book. I have some order forms with me’ (A second course in
languages and automata theory), and showing a picture of Frobenius (there is no paper by him). It was a brilliant talk on relations between numbers and words, as in the Sylvester problem, Shell sort, local and global stamp postage problem, and applications of the Frobenius problem.

Flavio d’Alessandro (co-author Stefano Varricchio), with *Well Quasi-orders in Formal Language Theory*, gave a good and nice fifth one to the memory of his co-author, also showing a picture of him. The presentation was on a generalized Nerode theorem, well quasi-ordered and unitary grammars, well quasi-order and shuffle closure of finite languages, and the Higman theorem. At the end he talked about the scientific work of Stefano Varricchio.

Zoltán Ésik, with the sixth invited talk *Iteration Semirings* gave a good and interesting survey on iteration semirings in the axiomatization of regular languages and rational power series, on ordered, continuous, inductive and other semirings.

Also to mention are the good and interesting presentations by Anil Ada on nondeterministic communication complexity of regular languages, stating that it is the ‘Swiss knife of complexity’, by Alexander Okhotin on state complexity of operations of finite FA’s, by Ashish Rastogi on testing ambiguity of FA’s, by Jens Glückler on deterministic forgetting automata, by Klaus Wagner on topological properties of regular languages, using Borel hierarchy, and by Flavio d’Alessandro on the Ćerný conjecture and transitive automata.

Other good talks were given by Ingo Meinecke on various constructions of tree automata from regular expressions, by Dominik Freydenberger on *bad news* on pattern decision problems, by Rodrigo de Souza on a new proof of decidability of the equivalence for transducers, by Szilárd Zsolt Fazekas on new bounds of powers of strings, by Aleksi Saarela on a new proof of Hmelevskii’s theorem, reducing the former one from 60 to 35 pages, and by Jana Hadraňová on large simple binary equality words, giving a ‘not very simple definition of simple words’ and finishing with ‘we are heading in a good direction’.

Nice and good presentations were also given by Michelangelo Bucci on a generalization of Episturmian words, by Marius Zimand on independent sequences, entropy, information and Kolmogorov complexity, by Štěpán Holub on the Ehrenfeucht-Silberger problem, by Frank Stephan on universal recursively enumerable sets of strings, by Ondřej Klíma on varieties of piecewise testable languages, and by Markus Holzer on shorter regular expressions for DFA’s.

Other good and interesting presentations were given by Carlo Mereghetti on desiritional complexity of regular languages, by Pawel Gawrychowski on growth functions of regular and context-free languages, by Andrei Romashchenko on aperiodic tiling, by Henning Börm on parallel communicating FA’s, and by Tobias Mömke on size complexity of rotating and sweeping automata.

Very good and interesting talks were given by Tomi Kärki on strongly and weakly periodic sequences, asking ‘What would be a good definition of a rela-
tional complexity function?’, and by Roberto Mantaci, wearing a Japanese head-
band like Kamikaze pilots but with ‘Success at examination’, on distribution of
squares in circular words.

The contributions of Erszébet Csuha-Vajk, Jurgen Dassow, György Vasził
and of Barbara F. Csima, Bakhadyr Khousainov were cancelled, and that of
Alberto Dennunzio, Enrico Formenti was presented by Andrei Romaschenko.

DLT 2008 was closed on Friday by Masami Ito. Volker Diekert informed us
on next DLT 2009 in Stuttgart, Germany, and Shinnooke Seki on DLT 2010 in
London, Ontario, Canada.

The proceedings of DLT 2008, edited by Masami Ito and Masafumi Toyama,
containing all invited talks and submissions, have been published as Springer
LNCS 5257.

In the breaks green tea, coffee, various soft drinks and Japanese snacks were
offered. Lunch was available in the two restaurants on KSU campus, offering
Japanese dishes.

Access to internet was available on 40 PC’s.

The social program started on Monday evening with a Welcome Reception in
Palace Side Hotel, near the Imperial Palace in Kyoto, with Japanese dishes, Kirin
beer, tea and coffee.

On Thursday afternoon we had a guided city tour with two buses to see some
parts of Eastern Kyoto. The first stop was at Nanzenji temple complex where
we visited the Zen style Hojo garden and could climb the 22 m high Sanmon
(Nakamon) gate. Because of being late the visit to the Kōdaiji temple complex
was skipped and we went immediately to the Kiyomizu-dera temple complex, founded
in 778, the present buildings in 1633 by Iemitsu Tokugawa. There in Kiyomizu
district we also had the possibility for buying typical Japanese souvenirs and food.

It was followed in the evening by the conference banquet in Palace Garden
Hotel, with a lot of Japanese dishes, beer, wine, Sake, tea and coffee. Addresses
were given by Masami Ito welcoming the participants, Kazuhiko Fukui, dean of
Faculty of Science of KSU, talking on DLT and research at KSU, Hidenosuke
Nishio, and Volker Diekert thanking the organizers.

Most participants stayed in the hotels Kyoto Garden Palace, Palace Side,
Holiday Inn Kyoto, Higashima Sanoyo, Kyoto Garden, ToyoKo Inn Kyoto Shijo-
Karasuma, between 30 and 60 minutes by public traffic from KSU, and in the
International House of KSU.

Weather was warm and humid, with highest temperatures between 25 and 30˚,
and with occasional light rain.

DLT 2008 was a successful conference, of high scientific level, and well orga-
nized in a nice atmosphere. Next DLT 2009 will be held at Stuttgart, Germany,
from June 30-July 3, 2009.
REPORT ON THE INTERNATIONAL WORKSHOP ON
THE COMPLEXITY OF SIMPLE PROGRAMS

Hector Zenil

The workshop main focus was on models of computation, computational complexity, and decidability. It was organised by Maureen Dwane, Niall Murphy, Turlough Neary, Anthony Karel Seda and Damien Woods at the The Boole Centre for Research in Informatics (BCRI), University College Cork, Ireland (where George Boole was once a researcher; UCC central library is named after him) on December 6th and 7th, 2008. The conference URL is http://www.bcri.ucc.ie/CSP08/

Among the speakers was Jack H. Lutz, whose talk entitled “A Divergence Formula for Randomness and Dimension” was about relating Hausdorff’s dimension to Kolmogorov complexity. There is a nice and fruitful connection between these 2 concepts capturing the complexity and the degree of randomness of an object, both approaches proven to be equivalent but each providing different insights of the same phenomena.

Philippe Moser’s talk entitled “A General Notion of Useful Information” proposed Bennett’s depth logic as a measure for useful information, an interesting and useful proposition. A nice exposition with good ideas and sensible results.

Gregory Lafitte’s talk entitled “Busy Beavers Gone Wild” was very interesting. It was about the complexity machines that print, not too much, but do a lot of work to print what they print. i.e. Busy Beavers machines in the sense of head movement rather than in the traditional sense of maximal number of printed non-blank symbols.

Klaus Sutner’s talk entitled “Computational Processes and Incompleteness” put Wolfram’s Principle of Computational Equivalence (PCE) in formal terms of intermediate constructive Turing degrees. He proposed a definition of physics-like computation and concluded that PCE is likely to be true in these terms. This is the case because basically intermediate Turing degrees are artificial constructions and actual physical computation seems to always follow a 0-1 law: it’s either universal or not. I have posted a more detailed account of this talk in http://www.animaexmachina.com/.

Cristian Calude’s talk (given remotely) entitled “Simplicity via Provability for Universal PreFx-free Turing Machines” was also very interesting. He proposed a
definition of simplicity based on whether the universality of a TM is provable on ZFC or PA. He showed that there are TMs that are provable either in ZFC or PA, and others that cannot be proven under ZFC or PA.

Jerome Durand-Lose’s talk entitled “Abstract Geometrical Computation 4: Small Turing-Universal Signal Machines” offered an interesting approach and new insights into his geometrical computation model through graphs.

Maurice Margenstern, a well known expert on the topics of small Turing machines and Cellular Automata on hyperbolic planes, gave a talk under the title “The Injectivity of the Global Function of a Cellular Automaton in the Hyperbolic Plane is Undecidable.” It included new and interesting results on the undecidability of some properties of hyperbolic Cellular Automata. A new very interesting paper from Margenstern is about to be published in the Complex Systems journal, a survey of the technique which allowed to give a simple proof that all 2-state 2-symbol Turing machines have a decidable halting problem, a result which was proved by L. Pavlotskaya in 1973.

Liesbeth De Mol’s interesting talk on Tag systems entitled “On the Boundaries of Solvability and Unsolvability in Tag Systems: Theoretical and Experimental Results” was particularly related to the problem that Emile Post himself tried to solve, and the similarities Collatz-sequence like problems.

Ivan Rapaport’s talk under the title “Communications in Cellular Automata” was very informative and clear, explaining the way one could compress the communication between two peers by means of sharing information as to the state and rule of a Cellular Automaton (CA). This can be also a characterisation of the behaviour and capabilities of CA.

There were several talks involving rule 110 CA and universality, namely Matthew Cook’s on “A Concrete View of Rule 110 Computation”, Nicolas Ollinger’s on “Intrinsically Universal Cellular Automata” and Manfred Kudlek’s titled “Some Considerations on Universality”. Cook presented a particular computation of rule 110 using the results obtained by Neary and Woods (simulating a Tag system in polynomial time).

Ollinger’s talk was basically about what he saw as the claim made in the NKS book that rule 110 collisions could be used to prove its intrinsic universality. He concluded that they cannot in reality be so used.

Leonid Levin gave a talk under the title “Simple Self-Organizing Networks”. As was to be expected, Levin was particularly active in the workshop, being one of the main founders of several seminal domains in the field of computer science, including the theory of algorithmic complexity and the theory of computational complexity that are at the core of the topics covered by the conference.

There was also a talk on research funds in Ireland. It seems they have a lot of money for the purpose, and they are trying to attract people (mainly foreigners) for postdocs and research exchange.
The Bulletin of the EATCS

I took advantage of the occasion to spread the word about the contest we are organizing: http://www.mathrix.org/experimentalAIT/TuringMachine.html and my own research (in joint with Jean-Paul Delahaye) on a stable definition of the algorithmic complexity independent of additive constants. As well as my work (in joint Cristian Calude) on the analysis of random number generators.

Successful submissions appeared in the pre-proceedings that were given during the workshop, published by Cork University Press. In addition, after the conference they announced a journal special issue in Theoretical Computer Science.

All in all it was a very fruitful meeting in a very nice atmosphere, and should be repeated often, there or elsewhere. A closely related conference "Machines, Universality and Computation" (MCU) conference, where many results and much research on small machines is presented, will be at Carnegie Mellon University (CMU) next year.
REPORT ON THE ACADEMIA EUROPAEA INFORMATICS SECTION WORKSHOP “GRAND CHALLENGES OF UNCONVENTIONAL COMPUTATION”

Igor Potapov

The Academia Europaea Informatics Section Workshop on Grand Challenges of Unconventional Computation was organized by Prof. Cristian Calude and Prof. Dines Bjorner in Liverpool, UK. The workshop was held at the University of Liverpool on September 17, 2008 in conjunction with the 2nd Workshop on Reachability Problems, 15-17 September 2008.

The workshop attracted more than thirty participants and had the outstanding list of speakers including Prof. Cristian S. Calude (U. Auckland), Prof. Jose Felix Costa (U. Wales Swansea), Prof. Dines Bjorner (Technical U. Denmark), Prof. Barry Cooper (U. Leeds), Prof. Mark Hogarth (U. Cambridge), Prof. Oscar Ibarra (UC. Santa Barbara) and Prof. John Tucker (U. Wales Swansea).

The workshop was opened by Prof. Cristian S. Calude, welcoming the participants and giving an interesting talk about the present and the future of Unconventional Computation discipline. The second talk was given by Prof. Dines Bjorner, the Chair of Informatics sections in The Academia Europaea. He also welcomed us, and spoke briefly on the history of Academia Europaea, recent initiatives of the organization and importance of the workshop’s subject for scientific community.

The first part of the workshop programme was continued by talks of Prof. Jose Felix Costa “The Misconception of Hyper-Computation” and Prof. Barry Cooper “Incomputability, Emergence, and the Turing Universe”. Prof. Jose Felix Costa started by discussing a procedure to locate the Equator by using a pendulum and pointing out that finding the absolute zero latitude is in this setting undecidable. The lesson taught by the “Gedanken experiment” is more general: information in Nature is degradable by measurements limitations. In the next talk Prof. Barry Cooper’s proposed to use computability as a key concept for understanding missing links between emergent higher mental functionality and algorithmic design. He also presented several arguments supporting the idea of a close relationship between emergence, the nature of phase transitions, and mathematical definability.

The second part of the programme was immediately started after the short refreshment break. The second part contained another three excellent talks, which
topics were ranging from concrete to very abstract by Prof. Mark Hogarth "A New Problem for Rule Following", Prof. Oscar Ibarra "Computing with Cells: Membrane Systems" and Prof. John Tucker “Computing by Experiments with Physical Systems”.

Prof. Mark Hogarth started his talk highlighting many analogies between developments in geometry and computability. There are many effects in non-classical computations that are similar to what we can observe in non-Euclidean geometries. One of the key points here is that “running” the same algorithm first on the Turing machine and then on the relativistic computers will, in general, produce two different results. Prof. Oscar Ibarra gave a nice brief overview of membrane computing and reported on recent results and challenging open problems in the field. He also discussed computational power and complexity of the recently introduced neural-like systems, called spiking neural P systems. The last talk of the workshop was given by Prof. John Tucker. In his talk he discussed the fundamental question: “What can we compute by experiments with a class of physical systems with or without using algorithms?” He proposed to explore experimental computation systematically and precisely starting from examples based on kinematics, possibly the simplest physical theory which offer insights into experimental computation, and pose interesting and difficult theoretical problems.

The workshop was very informative for most participants. It was very well organized and stimulated many fruitful discussions between the participants related to computability, mathematics and philosophy.
ABSTRACTS OF

PHD THeses
Abstract

The overall goal of this thesis is to provide new insights into the program semantics of concurrent programming languages. Since we are interested in the correctness of program transformations, we require a solid and canonical notion of equality of programs.

As a semantic model of a concurrent programming language we introduce the calculus $\Lambda_{let}^{amb}$ which is an untyped, higher-order, call-by-need lambda calculus extended by data constructors and case-expressions, explicit sharing and recursion via letrec-expressions, seq-expressions for modeling sequential computations, and the nondeterministic operator amb. This binary operator (proposed in 1961 by John McCarthy) is locally bottom-avoiding, i.e. if applied to two arguments $s$ and $t$, the operator chooses one of both arguments, but must not choose a divergent argument unless both arguments $s$ and $t$ are divergent.

Since well-known methods based on denotational models fail in the setting of nondeterminism including McCarthy’s amb-operator, the presented approach is purely based on operational semantics. The operational semantics of $\Lambda_{let}^{amb}$ is defined by a small-step reduction, called normal order reduction, which implements concurrent call-by-need evaluation. Since this reduction strategy is not fair, but fairness is required to ensure bottom-avoidance of amb, we also introduce a fair normal order reduction, which ensures fairness by using resources for the arguments of the amb-expressions for book-keeping.

Equality of programs is defined by using contextual equivalence, i.e. two expressions are equivalent iff they cannot be distinguished by observing their behavior in any program context. For the observed behavior it is not sufficient to take into account may-convergence, only, where may-convergence means the ability to reduce to a weak head normal form. Hence, additionally the must-convergence is observed, where an expression is must-convergent iff every successor w.r.t. nor-
normal order reductions is may-convergent. An analysis of the contextual order at the end of the thesis will show, that contextual equivalence can be simplified by taking into account must-convergence, only. Moreover, we show that our definition of must-convergence has some kind of fairness built-in, i.e. contextual equivalence defined by using normal order reduction is identical to contextual equivalence defined by using fair normal order reduction. This result considerably simplifies proofs dealing with contextual equivalence.

The main contribution of this thesis is to prove correctness of program transformations, where correctness means, that the transformation preserves contextual equivalence. For proving such correctness results we develop several proof techniques including a context lemma, a diagram based proof method, a standardization theorem, and a finite simulation method based on complete reduction successor sets and contextual set equivalence.

Using these techniques we show that all deterministic reduction rules of the calculus and some further commonly used program transformations are correct. Moreover, we show validity of some characteristic algebraic laws for the amb-operator and for encodings of other nondeterministic operators.

In the final part of the thesis we consider program transformations which translate programs from a higher-level language into a more lower-level language. Therefore a concurrent abstract machine for call-by-need evaluation together with a translation from $A_{\text{amb}}^{\text{let}}$-expressions into the machine language is introduced. We show correctness of this translation which implies that the concurrent abstract machine correctly evaluates $A_{\text{amb}}^{\text{let}}$-expressions.

Concluding, this thesis gives evidence, that the “operational approach” of using a contextual equivalence based on may- and must-convergence as program equality is successful for $A_{\text{amb}}^{\text{let}}$. Due to its generality this approach may be applied to various other concurrent programming calculi.

Table of Contents

1 Introduction and Background ......................................... 1

  1.1 Motivation and Survey ............................................. 1
  1.2 Results .......................................................... 21
  1.3 The Lambda Calculus and Extensions .............................. 24
  1.4 Outline .......................................................... 37

2 The Nondeterministic Call-by-Need Calculus $A_{\text{amb}}^{\text{let}}$ ........... 41

  2.1 The Language of $A_{\text{amb}}^{\text{let}}$ .................................. 43
  2.2 Operational Semantics ............................................. 48

184
The Bulletin of the EATCS

2.3 Convergence and Divergence .......................... 66
2.4 Fairness ............................................ 74
2.5 Encoding of Nondeterministic and Parallel Operators .... 86

3 Contextual Equivalence and Correctness of Program
Transformations ........................................... 93
  3.1 Contextual Equivalence ............................... 97
  3.2 Context Lemmas ..................................... 100
  3.3 Framework for Proving Correctness of Program Transformations 106
  3.4 Correctness of (lbeta) and (case-c) and (seq-c) .............. 118
  3.5 Additional Correct Program Transformations ................. 120
  3.6 Correctness of Deterministic Reduction Rules .............. 130
  3.7 Unique Copying .................................... 148

4 Standardization and Successor Set-Based Proof Technique ...... 161
  4.1 Normal Order Reduction is Standardizing ................. 165
  4.2 Analysis of Ω-Terms ................................ 174
  4.3 Proving Contextual Equivalence Based on Successor Sets ...... 188
  4.4 Analysis of the Contextual Preorders and Equivalences ....... 206
  4.5 Characteristic Laws of Nondeterministic Operators ........... 211

5 A Concurrent Abstract Machine ............................. 241
  5.1 The Calculus A_{let,vam}^- ................................ 244
  5.2 The Concurrent Abstract Machine CAM ..................... 262
  5.3 Correctness of CAM .................................. 279
  5.4 Correctness of the Fair Concurrent Abstract Machine .......... 306

6 Conclusion and Further Research ............................. 315


Author’s correspondence address  David Sabel
Institut für Informatik
Goethe-Universität
Postfach 11 19 32
D-60054 Frankfurt, Germany

Author’s email address  sabel@ki.cs.uni-frankfurt.de
Pictures from ICALP 2008, Reception, Excursion and Banquet  
(by M. Kudlek)

Matthew Hennessy, Mogens Nielsen, Adi Shamir

Sophie and Jean-Éric Pin, Erich Graedel, Juhani Karhumäki

Eric Badouel, Bruno Courcelle, Emanuela Morelli, Anna Ingólfsdóttir, Luca Aceto
BEATCS no 97

PICTURES FROM CONFERENCES

Jozef Gruska, Paul Spirakis

Mogens Nielsen, Josep Diaz, Kazuo Iwama

Martin Dietzfelbinger, Burkhard Monien, Friedhelm Meyer auf der Heide

Conference site

Crater Kerid

Geysir

188
The Bulletin of the EATCS

Geysir Strokkur

Troll or lost participant?

Gullfoss, Eike Best

Frontier between America and Europe

Þingvallavatn

A real computer scientist, Eike Best
The Bulletin of the EATCS

Pictures from DLT 2008
(by M. Kudlek)

Masami Ito, opening DLT 2008
Juraj Hromkovič
Victor Mitrana
Anil Ada
Toyo Sakai, president of KSU
Alexander Okhotin
BEATCS no. 97

PICTURES FROM CONFERENCES

Galina Jirásková  
Cyril Nicaud  
Zachary Kincaid

Ashish Rastogi  
Jens Glöckler  
Klaus Wagner

Flavio d’Alessandro  
Ingmar Meinecke  
Emmanuel Filiot
The Bulletin of the EATCS
BEATCS no 97

PICTURES FROM CONFERENCES

Rémi Morin  Aleksi Saarela  Tomi Kärki

Jana Hadrovová  Gerhard Lischke  Michelangelo Bucci

Yuji Kobayashi  Jeffrey Shallit  Marius Zimand

194
BEATCS no 97  PICTURES FROM CONFERENCES

Michael Luttenberger  Henning Bordihn  Tobias Mönke

Carlo Mereghetti  Hidenosuke Nishio, Martin Kutrib  Henning Bordihn, Ludwig Staiger

Masami Ito  closing DLT 2008  Volker Diekert  presenting DLT 2009  Shinnosuke Seki  presenting DLT 2010
The Bulletin of the EATCS
Pictures from AFLAS 2008
(by M. Kudlek)
The Bulletin of the EATCS

Sergey Afonin  Pál Dömösi  Bakhadyr Khoussainov

Zoltán Ésik  Gerhard Lischke  Manfred Kudlek

Géza Horváth  Marcin Jan Schroeder  Friedrich Otto
BEATCS no 97

PICTURES FROM CONFERENCES

Hidenosuke Nishio
Yoshiyuki Kunimochi
Benedek Nagy

Markus Lohrey
Peter Leupold
Libor Polák

Ondřej Klíma
Masafumi Toyama
Cristian Calude
The Bulletin of the EATCS

Szabolcs Iván

Hideki Tsuiji

Jian-Qi Liu

Masami Ito closing AFLAS

Go, Tatsuhiko Saito, Yuji Kobayashi, Kayoko, Tsuji, Shinnosuke Seki

Tatsuhiko Saito at barbecue
BEATCS no 97

Pictures from CS&P 2008
(by M. Kudlek)

1. Gabriela Lindemann-von Trzebiatowski
3. Ludwik Czaja

4. Manfred Kudlek
5. Hans Langmaack
6. Zbigniew Suraj

7. Berndt Farwer
8. Kamila Barylska
9. Jan Martijn van der Werf
The Bulletin of the EATCS

Irina Virbitskaite Jan-Thierry Wegener Wojciech Nabiałek

Jan Calta Lech Pocholski Lin-Anh Nguyên

Piotr Romanowski Hùng So-n Nguyên Beata Zielosko
BEATCS no 97

PICTURES FROM CONFERENCES

Marcin Szczuka
Roman Rydziejowski
Maciej Sreter

Krzystof Balicki
Bożena Woźna
Nadezhda Govedarova

Holger Schlingloff
Damas Gruska
Andrzej Zbrzezny

204
The Bulletin of the EATCS
Pictures from CSP 2008
(by M. Kudlek)

Anthony K. Seda
Patrick Fitzgerald
Maurice Margenstern

Matthew J. Patitz
Scott M. Summers
Grégory Lafitte

Klaus Suttner
Peter Kennedy
Liesbeth de Mol
The Bulletin of the EATCS

Conference site

Lunch, Martin Kutrib, Yurii Rogozhin, Sergei Verlan, Olivier Bournez

Dinner, Yurii Rogozhin, Klaus Suttner, Jérôme Durand-Lose, Alexander Okhotin, Maurice Margenstern, Liesbeth de Mol

Dinner, Fred Lunnon, Turlough Neary

Dinner, Damien Woods, Matthew Cook

Lunch, Olivier Bournez, Yurii Rogozhin
BEATCS no 97

PICTURES FROM CONFERENCES

Pub, Klaus Suttner, Jack Lutz, Alexander Okhotin

Pub, Turlough Neary, Klaus Suttner, Yurii Rogozhin, Sergei Verlan

Damien Woods serving beer

Post workshop meeting, Turlough Neary, Niall Murphy,
European Association for Theoretical Computer Science
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 International Colloquium on Automata, Languages and Programming (ICALP), the conference of EATCS.

MAJOR ACTIVITIES OF EATCS

- Organization of ICALP;
- Publication of the “Bulletin of the EATCS;”
- Award of research and academic careers prizes, including the “EATCS Award,” the “Gödel Prize” (with SIGACT) 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: ETAPS (The European Joint Conferences on Theory and Practice of Software), STACS (Symposium on Theoretical Aspects of Computer Science), MFCS (Mathematical Foundations of Computer Science), LICS (Logic in Computer Science), ESA (European Symposium on Algorithms), Conference on Structure in Complexity Theory, SPAA (Symposium on Parallel Algorithms and Architectures), Workshop on Graph Theoretic Concepts in Computer Science, International Conference on Application and Theory of Petri Nets, International Conference on Database Theory, Workshop on Graph Grammars and their Applications in Computer Science.

Benefits offered by EATCS include:

- Subscription to the “Bulletin of the EATCS;”
- 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;”

(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.
**The Bulletin of the EATCS**

**Sites of ICALP meetings:**
- Paris, France 1972
- Saarbrücken, Germany 1974
- Edinburgh, Great Britain 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, Great Britain 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
- Wroclaw, Poland 2007
- Reykjavik, Iceland 2008

(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;

- 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](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 series are W. Brauer (Munich), J. Hromkovic (Aachen), G. Rozenberg (Leiden), and 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 Editors-in-Chief of the journal currently are G. Ausiello (Rome), D. Sannella (Edinburgh), G. Rozenberg (Leiden), and M.W. Mislove (Tulane).

ADDITIONAL EATCS INFORMATION

For further information please visit http://www.eatcs.org, or contact the President of EATCS:
Prof. Dr. Giorgio Ausiello, Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”, Via Salaria 113, 00198 Rome, ITALY
Email: president@eatcs.org

EATCS MEMBERSHIP

DUES

The dues are €30 for a period of one year. 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 €25 per year. Additional €25 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, a subscription form can be downloaded from www.eatcs.org to be filled and sent together with the annual dues (or a multiple thereof, if membership for multiple years is required) to the Treasurer of EATCS:

Prof. Dr. Dirk Janssens, University of Antwerp, Dept. of Math. and Computer Science
Middelheimlaan 1, B-2020 Antwerpen, Belgium
Email: treasurer@eatcs.org, Tel: +32 3 2653904, Fax: +32 3 2653777

The dues can be paid (in order of preference) by VISA or EUROCARD/MASTERCARD credit card, by cheques, or convertible currency cash. Transfers of larger amounts may be made via the following bank account. Please, add €5 per transfer to cover bank charges, and send the necessary information (reason for the payment, name and address) to the treasurer.

Fortis Bank, Jules Moretuslei 229, B-2610 Wilrijk, Belgium
Account number: 220–0596350–30–01130
IBAN code: BE 15 2200 5963 5030, SWIFT code: GEBABE BB 18A

BEATCS no 97

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 Editors-in-Chief of the journal currently are G. Ausiello (Rome), D. Sannella (Edinburgh), G. Rozenberg (Leiden), and M.W. Mislove (Tulane).

ADDITIONAL EATCS INFORMATION

For further information please visit http://www.eatcs.org, or contact the President of EATCS:
Prof. Dr. Giorgio Ausiello, Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”, Via Salaria 113, 00198 Rome, ITALY
Email: president@eatcs.org

EATCS MEMBERSHIP

DUES

The dues are €30 for a period of one year. 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 €25 per year. Additional €25 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, a subscription form can be downloaded from www.eatcs.org to be filled and sent together with the annual dues (or a multiple thereof, if membership for multiple years is required) to the Treasurer of EATCS:

Prof. Dr. Dirk Janssens, University of Antwerp, Dept. of Math. and Computer Science
Middelheimlaan 1, B-2020 Antwerpen, Belgium
Email: treasurer@eatcs.org, Tel: +32 3 2653904, Fax: +32 3 2653777

The dues can be paid (in order of preference) by VISA or EUROCARD/MASTERCARD credit card, by cheques, or convertible currency cash. Transfers of larger amounts may be made via the following bank account. Please, add €5 per transfer to cover bank charges, and send the necessary information (reason for the payment, name and address) to the treasurer.

Fortis Bank, Jules Moretuslei 229, B-2610 Wilrijk, Belgium
Account number: 220–0596350–30–01130
IBAN code: BE 15 2200 5963 5030, SWIFT code: GEBABE BB 18A

BEATCS no 97

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 Editors-in-Chief of the journal currently are G. Ausiello (Rome), D. Sannella (Edinburgh), G. Rozenberg (Leiden), and M.W. Mislove (Tulane).

ADDITIONAL EATCS INFORMATION

For further information please visit http://www.eatcs.org, or contact the President of EATCS:
Prof. Dr. Giorgio Ausiello, Dipartimento di Informatica e Sistemistica
Università di Roma “La Sapienza”, Via Salaria 113, 00198 Rome, ITALY
Email: president@eatcs.org

EATCS MEMBERSHIP

DUES

The dues are €30 for a period of one year. 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 €25 per year. Additional €25 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, a subscription form can be downloaded from www.eatcs.org to be filled and sent together with the annual dues (or a multiple thereof, if membership for multiple years is required) to the Treasurer of EATCS:

Prof. Dr. Dirk Janssens, University of Antwerp, Dept. of Math. and Computer Science
Middelheimlaan 1, B-2020 Antwerpen, Belgium
Email: treasurer@eatcs.org, Tel: +32 3 2653904, Fax: +32 3 2653777

The dues can be paid (in order of preference) by VISA or EUROCARD/MASTERCARD credit card, by cheques, or convertible currency cash. Transfers of larger amounts may be made via the following bank account. Please, add €5 per transfer to cover bank charges, and send the necessary information (reason for the payment, name and address) to the treasurer.

Fortis Bank, Jules Moretuslei 229, B-2610 Wilrijk, Belgium
Account number: 220–0596350–30–01130
IBAN code: BE 15 2200 5963 5030, SWIFT code: GEBABE BB 18A