Bulletin
of the
European Association for
Theoretical Computer Science

EATCS

Number 99  October 2009
COUNCIL OF THE
EUROPEAN ASSOCIATION FOR
THEORETICAL COMPUTER SCIENCE

President: Burkhard Monien Germany
Vice Presidents: Giorgio Ausiello Italy
                Don Sannella United Kingdom
                Paul Spirakis Greece
Treasurer: Dirk Janssens Belgium
Bulletin Editor: Maria Serna Spain

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 Vladimiro Sassone United Kingdom
Fedor Fomin Norway Jiří Sgall Czech Republic
Giuseppe F. Italiano Italy Andrzej Tarlecki Poland
Juha Karhumäki 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 ....................................... bmon@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
The bulletin is entirely typeset by \texttt{PDF\LaTeX} and \texttt{Con\LaTeX} in \texttt{TXfonts}.

All contributions are to be sent electronically to

bulletin@eatcs.org

and must be prepared in \LaTeX\texttt{2\epsilon} using the class beatcs.cls (a version of the standard \LaTeX\texttt{2\epsilon} 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. 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 former President ........................................... 3
- Letter from the President .................................................. 5
- Letter from the Editor ...................................................... 8
- Report from the EATCS General Assembly ................................. 9
- The Gödel Prize 2010 ....................................................... 17
- The Presburger Award 2010 ................................................ 21

**Institutional Sponsors**
- EATCS sponsors ............................................................... 25

**EATCS News**
- The Japanese Chapter, by K. Makino ..................................... 29
- News from India, by M. Mukund ........................................... 31
- News from Latin America, by A. Viola .................................. 35
- News from New Zealand, by C.S. Calude ................................. 37

**The EATCS Columns**
- The Complexity Column, by J. Torán
  - Progress on Polynomial Identity Testing, by N. Saxena .......... 49
- The Concurrency Column, by L. Aceto
  - Algebraic Properties for Free!, by L. Aceto, A. Ingolfsdottir,
    M.R. Mousavi, M.A. Reniers .......................................... 81
- The Distributed Computing Column, by P. Fatourou
  - Composable Error Recovery With Transactional Memory, by
    T. Riegel, P. Felber, C. Fetzer ...................................... 105
- The Formal Specification Column, by H. Ehrig
  - Visual Modelling and Analysis of Model Transformations
    based on Graph Transformation, by C. Ermel ...................... 135
- The Natural Computing Column, by G. Rozenberg
  - Quantum Pseudo-Telepathy Saves the World, by M. Main ........ 153
THE PUZZLE CORNER, by L. Rosaz ............................... 169

REPORTS FROM CONFERENCES

The 36th International Colloquium on Automata, Languages and Programming, by M. Kudlek ................................. 173
20th Annual Symposium on Combinatorial Pattern Matching (CPM 2009), by T. Gagie ........................................ 183
The 11th Annual Workshop of Descriptional Complexity of Formal Systems (DCFS 2009), by I. McQuillan ................. 185
The International Computer Science Symposium in Russia (CSR 2009), by E.A. Hirsch ........................................ 188
The 35th International Workshop on Graph Theoretic Concepts in Computer Science (WG 2009), by H. Broersma .......... 192

EATCS LEAFLET .................................................... 195
EATCS MATTERS
Dear EATCS members,

As you will read in my report on the General Assembly that was held in Rhodes during ICALP 2009, at the Council meeting I have expressed my wish to step down from my office as EATCS President. The Council has elected Burkhard Monien as new President. Burkhard is one of the leading figures of theoretical computer science in Europe and I am sure he will put his excellent scientific and organizational experience at the service of EATCS and will provide an outstanding guidance to our Association through the future years. I wish him a successful activity.

It has been an exciting and challenging experience for me to lead EATCS in the last three years and I want to thank all the people that, in various roles, have given their advice and their support to my activity. During this period ICALP has continued to grow as one of the top world conferences in theoretical computer science and has attracted an unprecedented number of papers and attendees. Also a number of changes in the organization of EATCS have taken place. We have established a permanent and professional secretarial structure and we have moved the first steps toward expanding the scientific and 'political' role of the Bulletin. Also the role of EATCS has grown in the community through the strengthening of the relationship with the main European conferences in theoretical computer science, the promotion of best paper awards in the most relevant conferences in the field and the creation of a new prize, the
Presburger Award, intended to give recognition and visibility to young brilliant researchers.
I look forward to contributing to the success of EATCS also in the future in my new office of Vice President.

Giorgio Ausiello, Rome
September 2009
Letter from the President

Dear EATCS members,

during the last ICALP in Rhodes I had the honour of being elected as the new EATCS president. First, let me thank the past president, Giorgio Ausiello, for the excellent work he has done during the last three years. As a vice chair, I had the chance to get a deeper view into the multiple activities done by the president, the council and many other EATCS members for the benefit of our association. I am sure it will not be an easy task to be the successor of Giorgio Ausiello. But I will do my best. I am looking forward to this challenge and I hope you will be ready to extend the constant support you have given to Giorgio also to me.

ICALP 2009 was again a very successful conference. It was accompanied by the 4 workshops ALGOSENSORS (5th International Workshop on Algorithmic Aspects of Wireless Sensor Networks), DCM (5th International Workshop on Developments in Computational Models), FOCLASA (8th International Workshop on Foundations of Coordination Languages and Software Architectures), and QUANTLOG (Workshop on Quantitative Logics). We have to give thanks to the general chairs Paul Spirakis, Christos Kaklamanis and Elias Koutsoupias and the program chairs Susanne Albers (track A), Wolfgang Thomas (track B) and Alberto Marchetti-Spaccamela in cooperation with Yossi Matias (track C) for the perfect organization of the conference and the excellent choice of the program. The scientific quality was indeed remarkably high. The audience fully enjoyed invited talks by Georg Gottlob, Thomas Henzinger,
Kurt Mehlhorn, and Noam Nisan and on Wednesday afternoon a special session to the honour of Christos Papadimitriou (called "spiritual excursion" by Paul Spirakis) with the speakers Richard Karp, László Lovász, Noam Nisan, Tim Roughgarden, and Mihalis Yannakakis. The EATCS award was presented during the conference to Gérard Huet in recognition of his outstanding scientific contributions to theoretical computer science.

The organization of the next ICALP in Bordeaux is proceeding well. It will again be organized in three tracks. You can find the call for papers through the EATCS website. In Rhodes it was decided to hold ICALP 2011 in Zurich and ICALP 2012 in Warwick. The centennial of the birth of A. M. Turing is in 2012 and ICALP will be part of the celebration of this event.

Another important outcome of our meeting in Rhodes was the installation of a new award devoted to young researchers for outstanding contributions to theoretical computer science documented by a paper or a series of papers. The award is named after Mojzesz Presburger who accomplished his path breaking work on decidability of the theory of addition as a student in 1930.

All details on this new award including the call for nominations can be found on the EATCS website. At the next ICALP in Bordeaux the "Presburger Award" will be presented for the first time. In this context, I like to remind you that the calls for nominations for the EATCS Award 2010 and the Goedel Prize 2010 are still open until December 15th, 2009, respectively January 31st, 2010. Both Awards will be presented also at ICALP 2010.
The Bulletin of the EATCS

The electronic Council Election of new members has taken place in September. At the time of editorial deadline of this volume the analysis of the results is not available. On behalf of EATCS I like to offer our sincere thanks to all leaving council members for their commitment in the last four years!

As you know the Proceedings of ICALP 2009 was the first volume that appeared in the new subline of LNCS called ARCoSS (Advanced Research in Computing and Software Science). Nevertheless, the discussion regarding "open access publishing initiatives" is still ongoing and the council has approved the following procedure on this issue: The Publication Committee will run a poll among all EATCS members to get a more differentiated opinion picture on this issue. At this point, I like to invite you to participate in the vote as soon as it is available on the EATCS website.

Finally, all that remains for me is to encourage you to send your best paper to the next ICALP so that we can keep the standard as one of the world leading conference in theoretical computer science. See you in Bordeaux!

Burkhard Monien
October 2009
Letter from the Bulletin Editor

Dear Reader,

Welcome to the October 2009 issue of the Bulletin of the EATCS.

Besides the usual report on ICALP 2009 this issue contains a lot of material that will keep you informed about recent advances in TCS. The columns contain a lot of news about current research in TCS, while reports on conferences and news from Japan, Latin America, New Zealand and India will update you on less formal aspects of TCS.

As you will read in the report from the EATCS General Assembly 2009, we will try to find new ways to improve the Bulletin in the near future. In this direction it is my pleasure to announce that a new section in the form of "Letters to the editor" will be started on the next Bulletin, provided I get enough feedback from you! We are seeking short position papers (around one page) on developments of the community, social and organizational aspects, directions for teaching TCS, etc. Please, send your contribution to bulletin@eatcs.org.

I look forward to your reactions,

Maria Serna, Barcelona
October 2009
ICALP 2009
REPORT ON THE EATCS GENERAL ASSEMBLY 2009

The 2009 General Assembly (GA) of EATCS took place on Tuesday, July 7th, 2009, at the Rhodos Palace Hotel in Rhodes (Greece), site of ICALP 2009. The President Giorgio Ausiello opened the GA at 18:00.

The agenda consisted of the following items:

1) Report of the EATCS President
2) EATCS Offices
3) Council election 2009
4) Report on ICALP 2009
5) Report on ICALP 2010
6) Venue of ICALP 2011 and ICALP 2012
7) Relationship with other Societies
8) EU research matters
9) Questions, comments and suggestions

REPORT OF THE EATCS PRESIDENT. The President reported briefly on the EATCS activities between ICALP 2008 and ICALP 2009. He referred to the more detailed report posted a couple of weeks before the GA on the EATCS web page at www.eatcs.org. Item by item he also reported about Council decisions regarding various matters.

First of all, the President reported that during the last year 10 major scientific events took place under the auspices of EATCS. Besides EATCS has contributed to a variety of awards, namely, the Goedel Prize 2009, the EATCS Award 2009 and Best Paper Awards and Best Student Paper Awards at major conferences in theoretical computer science (ICALP, ETAPS, ESA, ICGT). Regarding publications, the President reminded that since the Fall 2008 the new Editor of the Bulletin is Maria Serna and since January 2009 a new column on Algorithmic Game Theory has been added to the Bulletin, edited by Marios Mavronicolas. The President also reported that, for the first time ICALP 2009 Proceedings have been published in the new LNCS Subline ARCoSS (Advanced Research in Computing and Software Science). Finally the President reported that in 2009 two more volumes have
been printed by Springer in the EATCS series of Texts and Monographs. Concerning financial matters the President reported that the financial situation of EATCS is good. EATCS had some benefits deriving from the savings in the mailing of the printed version of the Bulletin, from a more careful collection of membership fees and from positive results of financial investments. Then he also reported that for 2009 the Elsevier and the Springer sponsorships have been renewed and the renewal of the sponsorship by Microsoft Research Cambridge is about to be finalized. In 2008 an important development concerning the secretarial support has taken place. At the end of 2008 the Secretarial Office has been moved from Aarhus to Patras at CTI where Ioannis Chatzigiannakis has started his activity as new EATCS Secretary. Also EATCS membership lists, mailing lists and web site have been moved and reorganized in Patras. Currently EATCS has 724 members (not including ICALP 2009 participants). An automatic reminding system will help to follow the situation of members and will reduce the number of members that simply forget to renew their membership. The existing agreement with ESA and MFCS to offer EATCS membership in the full registration fee of these conferences to participants that are not already members of EATCS has also been an important development in view of increasing EATCS membership, especially in Central and Eastern Europe. For other sponsored conferences the Council has decided to ask the organizers to provide the option to register to EATCS in the registration page of the conference. In second place the Council has decided to introduce a 'student membership fee’ of 15 euros. A series of new services will also be implemented by the Secretarial Office in the future: support for forums, support for national chapters, support for electronic elections, information for members on EU matters, announcements of job positions.

Subsequently the President reported about EATCS activity concerning established awards and Council decisions regarding new awards. The Goedel Prize co-sponsored with ACM SIGACT has been assigned for the year 2009 to the following papers: (1) ”Entropy waves, the zig-zag graph product and new constant degree expanders”, by Omer Reingold, Salil Vadhan and Avi Wigderson, Annals of Mathematics, Vol 155, (2002), 157-187 and (2) ”Undirected connectivity in Log-Space”, by Omer Reingold, Journal of ACM, Vol 55 (4) (2007). The Committee consisted of Shafi Goldwasser (Chair), Johan Hastad, Cynthia Dwork and Colin Stirling, Jean Pierre Jouannaud, Mike Paterson. Since Colin Stirling’s term has expired the Council has decided to replace him with Mogens Nielsen. The EATCS Award 2009 has been assigned to Gerard Huet for his outstanding contributions to theoretical computer science and software science. The Committee consisted of Catuscia Palamidessi (Chair), Emo Welzl, Paul Spirakis. Since Catuscia Palamidessi’s term has expired the Council has decided to replace her with Eugenio Moggi. Concerning new awards initiatives the President reported that the EATCS Council has approved to create a new award devoted to young researchers.
for outstanding contributions documented by a paper (or a series of papers). The
award will be named after Mojzesz Presburger who accomplished his work on de-
cidability of the theory of addition (which today is called Presburger arithmetic)
as a student in 1930. Exceptionally the Award can be assigned to a group of re-
searchers. The recipient should be \( \leq 35 \) years at time of nomination. The award
will be presented each year at ICALP and will consist of reimbursement of travel
and lodging expenses for participation to ICALP plus 1000 euros. The deadline
for nominations should be December 31st. The Council has also decided that the
Presburger Award Committee for 2010 will consist of: Wolfgang Thomas (for
a term of one year; Chair), Andrzej Tarlecki (for a term of two years), Stefano
Leonardi (for a term of three years). The President reported that the Council has
also addressed other issues regarding awards (the creation of EATCS Fellows, the
creation of an EATCS Service Award, the creation of a Prize to be decided and
assigned in cooperation with ACM SIGLOG). These issues are left to the Award
Committee for further discussion. In conclusion the President informs the GA that
the Council has appointed the new Award Committee that will consist of J. van
Leeuwen, Chair, V. Diekert, J.P. Jouannaud, D. Peleg, V. Sassone, A. Tarlecki, J.
Wiedermann.

Finally the President reported to the GA about EATCS publications. The first
issue to be addressed concerned the Bulletin (BEATCS). In the Fall 2008 a poll has
been run by Publication Chair Luca Aceto on how the Bulletin may be improved.
According to the results of the poll the Bulletin readership would like to have: 1.
more scientific content (e.g. more surveys for non-specialists, book reviews, his-
tory of TCS, impact of TCS on practice accompanied by a broader understanding
of theory, more interviews and philosophical reflections) and 2. a more political
role (e.g. keeping abreast of developments in the community such as social and
organizational aspects, job opportunities, position papers on research assessment,
directions for teaching TCS etc.). As far as item 1 is concerned, the Council has
approved that the following regular items be added to the present contents of the
BEATCS:

- a new column devoted to applications of TCS, broadly construed;
- a new column devoted to contributions devoted to the history of TCS; and
- a book review section.

As far as item 2 is concerned, the Council has approved that a regular column
devoted to position papers on issues related to academic politics and to publication
and evaluation of research be added to BEATCS.

The Council has also decided that the EATCS web site will be enriched with
announcements of job positions and other announcements interesting members,
possibly in pages accessible only to EATCS members. Concerning distribution of the Bulletin the Council has decided to run a new poll in order to evaluate different options such as 1) continue with the current system, 2) eliminate completely the possibility to have a printed version of the Bulletin or at least charge the members who want to receive a printed copy for the mailing costs, 3) to print once a year a volume containing only the scientific part of the Bulletin. Upon request from the Council the President proposed the GA at this point to cast a straw vote to see how many people still want to receive a printed copy of the Bulletin. The result of the vote has been that the overwhelming majority was in favor to abolish the printed version of the Bulletin completely.

The second issue regarding publications addressed by the President concerned ICALP Proceedings. The President reported that following the discussions that took place in the Council last year in Iceland, and the consequent initiative that Vladimiro Sassone and himself took with Alfred Hofmann, the LNCS Publisher has decided to start a new 'Subline' of LNCS Sublibrary 1 (Theoretical Computer Science and General Issues). The title of the Subline is 'Advanced Research in Computing and Software Science - ARCoSS'. The Advisory Board consists of: S. Albers, X. Deng, B. Pierce, B. Steffen, M. Sudan, J. Wing. Editors of the series are: G. Ausiello, V. Sassone. The new subline will host the proceedings of high profile conferences in the fields of theory of computing and of formal approaches to software science, primarily (but not exclusively) EATCS conferences (such as ICALP and ESA) and ETAPS conferences (such as CC, ESOP, FASE, FOSSACS, TACAS). ICALP 2009 Proceedings are the first volumes that appear in the new 'subline'. ESA 2009 Proceedings will be the next. Further volumes will be printed subject to the decision of the Advisory Board. The initiative is inspired to similar initiatives taken by LNCS for other scientific domains. LNAI, LNBI are subseries devoted to Artificial Intelligence and to Bioinformatics respectively. Information Security and Cryptology is a sublibrary realized in cooperation with the International Association for Cryptological Research. The President also reported that a discussion with Springer has been started regarding the possibility to allow EATCS Members to freely access the electronic version of some of the volumes of the ARCoSS series. Similarly IACR provides free access to its members to some LNCS proceedings in the sublibrary Information Security and Cryptology (CRYPTO, EUROCRYPT, ASIACRYPT, FSE, CHES, PKC, and TCC). The Council has agreed that until ICALP Proceedings are published in the ARCoSS series it is natural to establish some sort of agreement with Springer for the free access to electronic versions of the Proceedings. It has been left open to decide to what conference proceedings the agreement should be extended.

The President also reported to the GA about Council decisions regarding the use of open access policy for EATCS publications. The Council has decided that - the Publications Committee will run a poll among EATCS members to understand
what are the feelings of members with respect to these issues - the Publications Committee will investigate on various aspects regarding electronic publishing initiatives; in particular the following points should be clarified: how various initiatives address the issue of quality and selectiveness, what guarantees they give about continuity, what are the overall (explicit and hidden) costs involved by various initiatives.

In conclusion the President informs the GA that the Council has confirmed the composition of the Publications Committee: L. Aceto, Chair, J. Diaz, J. Karhumaki, C. Palamidessi, V. Sassone, M. Serna.

EATCS OFFICES. The President reported that due to his decision to step down one year in advance with respect to the foreseen deadline, a Search Committee has been appointed consisting of the current President and the two most recent Past Presidents: Josep Diaz and Mogens Nielsen. The Committee has examined various possibility and has unanimously decided to propose to the Council the name of Burkhard Monien. During his meeting of July 6th the Council has unanimously elected Burkhard Monien as new EATCS President. The Council has also confirmed Don Sannella and Paul Spirakis as Vice-Presidents and has appointed Giorgio Ausiello as Vice-President for the period of one year in view of the support that he can give to the new President to carry on various ongoing initiatives of EATCS.

Giorgio Ausiello congratulated with the new President Burkhard Monien and expressed his sincere best wishes for a fruitful work. Also the President expressed his most sincere thanks to the Vice-Presidents, to all Council Members and to all Committees’ members for their contributions to the life of the Association and for the support and pleasant collaboration that they have provided to him. A special thank was addressed to the Treasurer Dirk Janssens, to the Bulletin Editor Maria Serna and to the Secretary Ioannis Chatzigiannakis and to all CTI people that support the activity of EATCS.

COUNCIL ELECTION 2009. The President reminded to the GA that in 2009 the term of the following members elected in 2005 expired: G. Ausiello (IT), L. Aceto (IS), G.F. Italiano (IT), E. Moggi (IT), C. Palamidessi (FR), D. Sannella (UK), J. Sgall (CZ), W. Thomas (DE), E. Welzl (CH). In September EATCS members will be asked to elect other ten Council members through an electronic vote. The President reported the list of nominations received before the GA. According to the decision taken at the Council in 2008 all nominees (who are willing to accept to be candidates) will be invited to post a brief statement of their views on EATCS in their web sites. The electronic ballot will be held in the period September 7th - October 6th. After brief discussion the GA approved the following list
REPORT ON ICALP 2009. First of all the General Co-Chairs of ICALP 2009 Spirakis, Kaklamanis and Koutsoupias presented their report on the organization of the conference. The GA expressed its appreciation for the excellent organization and the great work done by the General Chairs and their staff. The scientific program was divided in three tracks: Track A - Algorithms, Automata, Complexity and Games (PC Susan Albers), Track B - Logic, Semantics, and Theory of Programming (PC Chair Wolfgang Thomas), Track C - Foundations of networked computation (PC Co-Chair Alberto Marchetti-Spaccamela and Yossi Matias). The figures of submitted and accepted papers are as follows: Track A 223, 62, Track B 84, 24, Track C 61, 22. (the slightly higher rate of acceptance for Track C is due to the need to consolidate the track on Foundations of networked computation, that this year appears for the first time). The President asked then the PC Chairs to report some statistics about the different tracks. Details can be found in this issue of the Bulletin in the usual report contributed by Manfred Kudlek. The GA also expressed its appreciation for the excellent work done by the Program Committees. Giorgio Ausiello continued the tradition to present the ICALP organizers with small gifts thanking them for the great job they have done.

REPORT ON ICALP 2010. The President reminded that in 2010 ICALP will be held in Bordeaux. General Chairs will be Claude Kirchner and Cyril Gavoille (who apologize for not being present in Rhodes). The report on the organization of ICALP 2010 was presented to the GA by Igor Walukiewicz. Again in 2010 the conference will be organized in three tracks as ICALP 2009, respectively chaired by Paul Spirakis, Samson Abramsky and Friedhelm Meyer auf der Heide. The following six speakers have been invited: Pierre Fraigniaud, Jean Goubault-Larrecq, Burkhard Monien, Joel Ouaknine, Roger Wattenhofer, Emo Welzl. The Web site and the call for papers have been prepared and hardcopies of the CfP have been distributed during ICALP 2009.

VENUE OF ICALP 2011 AND ICALP 2012. The President reported that for ICALP 2011 only one bid has been put forward, by ETH, Zurich. The proposal was then presented to the GA. After the presentation the GA approved the organization of ICALP 2011 at ETH in Zurich.

The President then reminded that 2012 is the centennial of the birth of A. M. Turing. AT the GA 2008 it has been agreed that ICALP will be held in UK in the
framework of the celebration of such event. A bid from University of Warwick has been submitted. Due to the need to coordinate with all other events foreseen for the celebration, exceptionally the decision about the location of ICALP 2012 had to be taken in 2009 instead of 2010. The proposal was then presented to the General Assembly. After the presentation the GA approved the organization of ICALP 2012 at the University of Warwick.

RELATIONSHIP WITH OTHER SOCIETIES. The President first reported to the GA about relationships with ETAPS. In particular the President reported that currently the two EATCS Representatives in the ETAPS Steering Committee are: Hartmut Ehrig and Catuscia Palamidessi. Since Hartmut Ehrig has announced that he is willing to step down, Don Sannella has been nominated by the Council as new EATCS representative. Concerning relationship with AAAC the President reported that correspondence with AAAC President Kazuo Iwama concerning the possibility of an agreement for joint (AAAC-EATCS) membership has been carried on. The President expressed his hopes that a conclusion can be reached in the near future.

The President also reported about the initiative taken by the EU Commission to stimulate the creation of European Scientific and Professional Societies in the field of ICT capable to play a role similar to the role plaid by ACM and IEEE in US. The first meeting has been held on November 24th, 2008, in Lyon, the second will be held in Brussels on October 13th, 2009. To the meeting have taken part representatives of European scientific and professional societies and supranational institutions (EATCS, EuroGraphics, EuroSys, ICST, BCS, CEPIS, NESSI, Informatics Europe, ERCIM). The focus of the first meeting and of the subsequent activity carried on by the promoters has been the creation of a Federation of existing societies (in cooperation with ACM and national societies such as GI and BCS) to increase the role of ICT in Europe regarding: university and professional education, research, industrial policy, standards. The related activities are still in an exploratory stage. The President considers that in this framework EATCS policy should be inspired by the following guidelines: - follow the developments of the existing initiatives giving support only to those initiatives that provide the maximum scientific guarantee - in the meantime operate toward the federation of European scientific societies (EATCS, EACSL, EAPSL, EASST etc.) as a first step. After brief discussion the GA approved the guidelines.

EU RESEARCH MATTERS. The President invited Paul Spirakis to report about recent developments in EU research. At the end of the report the President thanked Paul Spirakis for his presentation. The President announced that the slides of SpirakisÕ report would have been posted in the ÔEU MattersÕ Section
of EATCS web site that may be accessed only by EATCS members.

**QUESTIONS, COMMENTS AND SUGGESTIONS.** At this point Giorgio Ausiello gave the floor to Manfred Kudlek who commented on the statistics of the authors who published repeatedly at ICALP and presented the special EATCS badge to those having reached 5 or more full papers at ICALP. According to the tradition Kudlek also presented the EATCS badges to the Editors of ICALP 2009 Proceedings.

At 20:30, since no other matter had to be addressed, the President thanked all present and closed the 2009 General Assembly of EATCS.

*Giorgio Ausiello*
Gödel Prize 2010

Call for Nominations


The Gödel Prize for outstanding papers in the area of theoretical computer science is sponsored jointly by the European Association for Theoretical Computer Science (EATCS) and the Association for Computing Machinery Special Interest Group on Algorithms and Computation Theory (ACM-SIGACT). This award is presented annually, with the presentation taking place alternately at the International Colloquium on Automata, Languages, and Programming (ICALP) and the ACM Symposium on Theory of Computing (STOC). The eighteenth presentation will take place during ICALP 2010, Bordeaux, France, July 5 to 12, 2010. The Prize is named in honor of Kurt Gödel in recognition of his major contributions to mathematical logic and of his interest, discovered in a letter he wrote to John von Neumann shortly before von Neumann’s death, in what has become the famous "P versus NP" question. The Prize includes an award of USD 5000.

Award Committee: The winner of the Prize is selected by a committee of six members. The EATCS President and the SIGACT Chair each appoint three members to the committee, to serve staggered three-year terms. The committee is chaired alternately by representatives of EATCS and SIGACT, with the 2010 Chair (Jean-Pierre Jouannaud) being an EATCS representative. The 2010 award committee consists of Cynthia Dwork (Microsoft Research), Johan Håstad (KTH, Stockholm), Jean-Pierre Jouannaud (INRIA and Tsinghua University), Mogens Nielsen (University of Aarhus), Mike Paterson (University of Warwick), and Eli Upfal (Brown University).

Eligibility: The last change of rules goes back to the 2005 Prize. The (parametric) rule can be found on websites of both SIGACT and EATCS. The rule for the 2010 Prize is given below and supersedes any different interpretation of the parametric rule.

Any research paper or series of papers by a single author or by a team of authors is deemed eligible if
(i) the paper was published in a recognized refereed journal no later than January 31st, 2010.
(ii) the main results were not published (in either preliminary or final form) in a journal or conference proceedings before January 1st, 1997.

The research nominated for the award should be in the area of theoretical computer science. The term "theoretical computer science" is meant in a broad sense, and encompasses, but is not restricted to, those areas covered by ICALP and STOC. Nominations are encouraged from the broadest spectrum of the theoretical computer science community so as to ensure that potential award-winning papers are not overlooked. The Award Committee shall have the ultimate authority to decide whether a particular paper is eligible for the Prize.

NOMINATIONS: Nominations for the award should be submitted by email to the Award Committee Chair at the following address:

jeanpierre.jouannaud@gmail.com

It is the duty of the Award Committee to actively solicit nominations. To be considered, nominations for the 2010 prize must be received by January 31st, 2010. Nominations may be made by any member of the scientific community. A nomination should contain a brief summary of the technical content of the paper and a brief explanation of its significance. A printable copy of the research paper or papers should accompany the nomination. The nomination must state the date and venue of the first conference or workshop publication or state that no such publication has occurred. The work may be in any language. However, if it is not in English, a more extended summary written in English should be enclosed. Additional recommendations in favor of the nominated work may also be enclosed. To be considered for the award, the paper or series of papers must be recommended by at least two individuals, either in the form of two distinct nominations or one nomination including recommendations from two different people. Those intending to submit a nomination are encouraged to contact the Award Committee Chair by email well in advance. The Award Committee will accept informal proposals of potential nominees, as well as tentative offers to prepare formal nominations. The "Subject" line of all related messages should begin with "Goedel2010".

SELECTION PROCESS: Although the Award Committee is encouraged to consult with the theoretical computer science community at large, the Award Committee is solely responsible for the selection of the winner of the award. The prize may be shared by more than one paper or series of papers, and the Award Committee reserves the right to declare no winner at all. All matters relating to the selection process that are not specified here are left to the discretion of the Award Committee.
The Bulletin of the EATCS

PAST WINNERS:


1999: Peter W. Shor, “Polynomial-time algorithms for prime factorization and
discrete logarithms on a quantum computer,” SIAM Journal on Computing

1998: Seinosuke Toda, “PP is as hard as the polynomial-time hierarchy,” SIAM

1997: Joseph Halpern and Yoram Moses, “Knowledge and common knowledge

1996: Alistair Sinclair and Mark Jerrum, “Approximate counting uniform genera-
tion and rapidly mixing Markov chains,” Information and Computation
82 (1989), 93–133.


1995: Neil Immerman, “Nondeterministic space is closed under complementa-

Róbert Szelepcsényi, “The method of forced enumeration for nondeter-

1994: Johan Håstad, “Almost optimal lower bounds for small depth circuits,”

proof system and a hierarchy of complexity classes,” Journal of Computer

Shafi Goldwasser, Silvio Micali and Charles Rackoff, “The knowledge
complexity of interactive proof systems,” SIAM Journal on Computing 18
Starting in 2010, the European Association of Theoretical Computer Science (EATCS) confers each year at the conference ICALP the

**Presburger Award**

to a young scientist (in exceptional cases to several young scientists) for outstanding contributions in theoretical computer science, documented by a published paper or a series of published papers.

The award is named after Mojzesz Presburger who accomplished his path-breaking work on decidability of the theory of addition (which today is called Presburger arithmetic) as a student in 1929.

Nominations for the Presburger Award can be submitted by any member or group of members of the theoretical computer science community except the nominee and his/her advisors for the master thesis and the doctoral dissertation. Nominated scientists have to be at most 35 years at the time of the deadline of nomination (i.e., for the Presburger Award of 2010 the date of birth should be in 1974 or later).

The Presburger Award Committee of 2010 consists of Stefano Leonardi (Rome), Andrzej Tarlecki (Warsaw), and Wolfgang Thomas (Aachen, chair). Nominations, consisting of a two page justification and (links to) the respective papers, should be sent to Wolfgang Thomas, RWTH Aachen, Informatik 7, 52056 Aachen, Germany or: thomas@informatik.rwth-aachen.de

by 31st December 2009.

The award includes an amount of 1000 â€” and an invitation to ICALP 2010 for a lecture.

EATCS gratefully acknowledges the financial support of Bertinoro International Center for Informatics to the Presburger Award.
INSTITUTIONAL SPONSORS
BiCi, Bertinoro international Center for informatics
   Bertinoro, Italy

BRICS, Basic Research in Computer Science
   Aarhus, Denmark

CTI, Computer Technology Institute
   Greece

Elsevier
   Amsterdam, The Netherlands

Microsoft Research
   Cambridge, United Kingdom

Springer-Verlag
   Heidelberg, Germany
EATCS News
REPORT FROM THE JAPANESE CHAPTER

K. Makino (Univ. of Tokyo)

EATCS-JP/LA Workshop on TCS
The eighth EATCS/LA Workshop on Theoretical Computer Science will be held at Department of Mathematics, Kyoto Univ., Feb. 1 ∼ 3, 2010. 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.

A formal call for papers will be announced at our web page early November, and a program will be announce early January, where we are also planning to announce a program in the next issue of the Bulletin. Please check our web page around from time to time. If you happen to stay in Japan around that period, it is worth attending. No registration is necessary for just listening to the talks; you can freely come into the conference room. (Contact us by the end of November if you are considering to present a paper.) Please visit Kyoto in its most beautiful time of the year!

7th EATCS-JP/LA Presentation Award
The seventh EATCS/LA Workshop on Theoretical Computer Science was held at Research Institute of Mathematical Sciences, Kyoto Univ., Feb. 2 ∼ 4, 2009. Dr. Ryuhei Uehara (Japan Advanced Institute of Science and Technology (JAIST)) who presented the following paper, was selected as the 7th EATCS/LA Presentation Award.

Complexity of Pleats Folding
by Tsuyoshi Ito, Masashi Kiyomi, Shinji Imahori, and Ryuhei Uehara

The award was given to him at the Summer LA Symposium held in August 2009. Congratulations! Please check our web page for the detail information and the list of presented papers.

On TCS Related Activities in Japan:
TGCOMP Meetings, January ∼ June, 2009
The IEICE, Institute for Electronics, Information and Communication Engineers of Japan, has a technical committee called TGCOMP, Technical Group on foundation of COMPuting. During January ∼ June of 2009, TGCOMP organized 4 meetings and 33 papers (including two tutorials) were presented there. Topics presented are, very roughly, classified as follows.
Algorithm: On Graphs (7)            Combinatorics / Probabilistic Analysis (4)
Algorithm: On Strings (3)           Computational Complexity (2)
Algorithm: On Other Objects (4)     Cryptography (1)
Data Structure (1)                  Distributed Computing (2)
Counting and Enumeration (5)        Quantum Computing (4)

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

---------

THE JAPANESE CHAPTER

CHAIR:  KAZUO IWAMA
VICE CHAIR: OSAMU WATANABE
SECRETARY: KAZUHISA MAKINO
EMAIL:  eatcs-jp@is.titech.ac.jp
URL:    HTTP://WWW.MISOJIRO.T.U-TOKYO.AC.JP/EATCS-J/INDEX.HTML

---------
News from India

by

Madhavan Mukund
Chennai Mathematical Institute
Chennai, India
madhavan@cmi.ac.in

In this edition of News from India, we review and preview some past and forthcoming meetings.

Formal Methods Update 2009

For the past few years, the Indian Association for Research in Computing Science (IARCS) has organized regular “update” meetings in the area of formal methods. These meetings are intended as a forum for Indian researchers and students to update themselves on current trends and to explore new research areas.

This year’s event was organized by Rajdeep Niyogi at the Indian Institute of Technology, Roorkee during July 13–15, 2009. The organization was impeccable. The campus is an interesting and harmonious mix of old and new buildings — IIT Roorkee has its origins in the Thomson Engineering College, the first engineering college set up in India, over 150 years ago.

The meeting was held over two and a half days, with surveys on topics such as solution concepts in games, model-checking over event structures, reachability in Petri nets and the relationship between existential monadic second-order logic and communicating automata.

The turnout was lower than expected, perhaps because of the relative inaccessibility of the venue. Roorkee is in the foothills of the Himalayas, about 5 hours from Delhi by road or train. Nevertheless, for those who did make it, it was a fruitful meeting.

The venue for the update meeting in 2010 has tentatively been fixed as the Indian Institute of Science, Bangalore. More details will be announced via the webpage http://fmindia.cmi.ac.in.
FSTTCS 2009

The 29th edition of FSTTCS, the annual conference of the Indian Association for Research in Computing Science (IARCS) will be held at the Indian Institute of Technology, Kanpur in December, 2009. The conference forms part of the golden jubilee celebrations of IIT Kanpur. The Programme Committee for FSTTCS 2009 is chaired by Ravi Kannan (Microsoft Research, Bangalore) and K. Narayan Kumar (Chennai Mathematical Institute).

The invited speakers for FSTTCS 2009 are Anuj Dawar (Cambridge), Kim G. Larsen (Aalborg), Martin Odersky (Lausanne), R. Ravi (CMU) and Avi Wigderson (Princeton).

The list of accepted papers is now available at http://www.fsttcs.org and registration will open soon. There will be two workshops organized alongside the conference, on game theory and streaming algorithms.

TECS Week 2010

The 8th TCS Excellence in Computer Science Week (TECS Week 2010) will be held at the Tata Research Development and Design Centre (TRDDC), Pune from 4 to 8 January, 2010. TECS Week is conducted by TRDDC jointly with United Nations University (UNU/IIST) and the Indian Association for Research in Computing Science (IARCS).

The theme for next January’s TECS Week is Formal methods in software verification, testing and debugging. The speakers are Patrice Godefroid (Microsoft Research, Redmond, USA), Daniel Kröning (Oxford Univ, UK), Rupak Majumdar (UCLA, USA), and Natarajan Shankar (SRI International, USA).

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


ISLA 2010

Since 2006, the logic community in India has been conducting an annual series of conferences, workshops and instructional schools each January. The community is now formally organized as the Association for Logic in India (ALI). In January 2009, ALI organized the Indian Conference on Logic and Applications (ICLA) in Chennai. The general plan is to alternate conferences and instructional schools. With this in mind, ALI’s next meeting will be the Indian School on Logic and Applications (ISLA), to be held in Hyderabad in January, 2010. Vi-
neet Padmanabhan (University of Hyderabad) will chair the organization. Mihir Chakraborty (Calcutta University) and Kamal Lodaya (Institute of Mathematical Sciences) chair the programme committee.

The ALI webpage is at http://ali.cmi.ac.in.

Madhavan Mukund, Chennai Mathematical Institute
Secretary, IARCS (Indian Association for Research in Computing Science)
http://www.cmi.ac.in/~madhavan
This issue is devoted to Imre Simon who passed away on August 12, 2009, in Sao Paulo, Brazil. We also present the call for participation to LAGOS 2009 to be held in Gramado, Brazil.

**Imre Simon**

(This note is a tribute to Imre Simon by his co-workers and former students.)

Imre Simon died on August 12, 2009, in Sao Paulo, Brazil. He would have turned 66 two days later.

Imre was one of the pioneers of Computer Science in Brazil and arguably in South America. He made significant scientific contributions to the theory of computing and exercised a natural academic, scientific and practical leadership.

Imre was a Full Professor (retired) in the Department of Computer Science of the Institute of Mathematics and Statistics of the University of Sao Paulo. He was a member of the Brazilian Academy of Sciences and played a decisive role in the committees of various funding agencies in Brazil.

In 1989, he was awarded the UAP Prize in France (along with M. Gromov and J.E. Stiglitz). In 1992, he organized the first symposium of the LATIN (Latin American Theoretical Informatics) series.

Professor Simon is survived by his wife, Gabriella Simon, and three children and four grandchildren.

Call for participation: LAGOS 2009

LAGOS (Latin-American Algorithms, Graphs and Optimization Symposium), is the union of two Latin American Conferences on these subjects: the GRACO Â˚ Brazilian Symposium on Graphs, Algorithms and Combinatorics and the LACGA Â˚ Latin American Conference on Combinatorics, Graphs and Applications.

LAGOS 2009 will be held at the Serra Azul Hotel, in the city of Gramado, Brazil. Gramado is a small city (30,000 habitants) 118 km (74 miles) away from Porto Alegre, the capital of the state of Rio Grande do Sul. Most of the population of Gramado is of German or Italian descent and its architecture and gastronomy reflects that. Gramado hosts the major South American film festival and is also known for its natural landscapes and the beautiful hydrangeas blossoming in late spring.

Regional Events

News from New Zealand
by
C. S. Calude

Department of Computer Science, University of Auckland
Auckland, New Zealand
cristian@cs.auckland.ac.nz

1 Scientific and Community News

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


2 A Dialogue about Constructive Mathematics with Professor Douglas Bridges

Professor Douglas Bridges has authored 160 research papers, and 7 books, on constructive mathematics and mathematical economics. The books include Constructive Analysis (with Errett Bishop) and Varieties of Constructive Mathematics (with Fred Richman), which together were the definitive reference works on
(Bishop-style) constructive mathematics for over 20 years. The latest book, Techniques of Constructive Analysis (with Luminţă Vîţă), is the only exposition of the major developments in constructive analysis since the previous two books appeared. He has been Chairman/Head of Department in three universities, including the University of Canterbury, where he has been Professor of Pure Mathematics since 1999. He is a former President of the New Zealand Mathematical Society. Professor Bridges has been awarded many distinctions: D.Sc. by Oxford University (2000), Fellow of the Royal Society of New Zealand (2000), Corresponding Fellow of the Royal Society of Edinburgh (2004). He has been a visiting Professor in several institutions, including New Mexico State University (1979 and 1988), Cornell University (1993), and Ludwig-Maximilians-University of Munich (2003, 2009, 2010). He is a member of a consortium of mathematicians from Europe, Japan and Christchurch who successfully bid for an IRSES Marie Curie Award for 2009-10 from the European Union.

Cristian Calude: Tell us a short history of constructive mathematics.

Douglas Bridges: Although the origins of constructive mathematics can be traced to people like Kronecker in the late 19th century, the subject made its grand entry on the mathematical stage when L.E.J. Brouwer (famed also for his contributions to classical topology) published his doctoral thesis “On the Foundations of Mathematics” in Amsterdam in 1907. In the thesis, Brouwer expounded his mathematical philosophy of “Intuitionism”, in which mathematical objects are creations of the human mind rather than objects in some extra-mental universe. A consequence of Brouwer’s philosophy is that, in proving the existence of some mathematical object, the intuitionist has to show, at least in principle, how that object can be constructed/computed. This process contrasts sharply with the pervasive mathematical practice of proving that some object \( x \) exists by first assuming that it doesn’t, then deriving (by correct logical deductions) a contradiction, and finally concluding that the object had to exist after all; that process does not enable you to construct the desired object.

Brouwer’s approach to mathematics was highly controversial, and eventually resulted in a major breach between the small group of intuitionistic mathematicians and the dominant group, led by the enormously talented and influential David Hilbert, of “classical” mathematicians, for whom Brouwer’s constructive methods were too restrictive. Since very few mathematicians took up Brouwer’s ideas in their daily mathematical practice (though some—notably Hermann Weyl—were sympathetic to his views), it appeared for some decades that, as Bourbaki famously claimed, the intuitionistic school would be a “memory ... no doubt destined to remain only as an historical curiosity” [2].

In the late 1940s in the Soviet Union, A.A. Markov developed an alternative form of constructive mathematics in which, rather than using Brouwer’s special
intuitionistic principles ("continuity principle" and "fan theorem"), one worked with recursive functions and used intuitionistic logic (a logic that had been abstracted from intuitionistic mathematical practice). Markov’s school of recursive constructive mathematics (RUSS) had some successes, but the restriction to recursive functions was also seen by many as a major hurdle to its acceptance in general.

Things changed considerably from 1967 onwards. In that year, Errett Bishop, a young American mathematician who had already established a formidable reputation for his research in functional analysis and several complex variables, published *Foundations of Constructive Analysis*, in which, single-handedly, he developed constructive treatments of large parts of real, complex, and functional analysis, including measure theory. Initially, only a few mathematicians took up Bishop’s challenge to work in Bishop-style constructive mathematics (BISH, which in practice is mathematics carried out with intuitionistic logic and an appropriate set-theoretic foundation, but without any of Brouwer’s principles or the Markovian restriction to a recursive-function-theoretical framework). However, the rise of computer science, with the resulting increase in awareness of issues of computability/constructibility, gradually created growing interest in BISH, and since 2000 there have been several conferences largely devoted to constructive mathematics and related issues.

**CC:** How relevant is philosophy in constructive mathematics?

**DB:** It is relevant to one rationale for working constructively: namely, that a subscriber to Brouwer’s intuitionistic philosophy or something close to it would regard a constructive approach as the only justifiable, meaningful one for mathematics. It is less relevant to most constructive mathematical practice, in the sense that even a Platonist might appreciate that the use of intuitionistic logic (and an appropriate set theory) is a good idea if one wants to distinguish between constructive and nonconstructive aspects of mathematics. Such a distinction is a meaningful one. Moreover, Bishop-style constructive proofs come with a bonus: not only can implementable algorithms be extracted from them, but the proofs are themselves proofs that those algorithms are correct (meet their specifications). You don’t need to be a committed philosophical constructivist to see the merit in such a proof; but you do need to be prepared to work with intuitionistic logic if you want to get your hands on one.

**CC:** Constructive mathematics is particularly relevant in computer science. Can you elaborate on this issue.

**DB:** I’ve really done this in my previous answer. Constructive proofs contain algorithms that can be (and in many cases have been) extracted and implemented, the proofs also showing the correctness of the extracted algorithms. This has
considerable significance for theorem-proving systems and the like, several of which have been created over the past quarter-century.

CC: Can you illustrate the role of constructive mathematics for theorem-proving systems?

DB: This is an area about which I am woefully ignorant. However, although I understand that one can use classical logic in proof assistants, it seems to me that using constructive logic is the right thing to do when one is interested in algorithmic proofs. For example, what would be the point of, say, using classical logic to prove the correctness of an implemented constructive proof, when that proof already embodies a constructive correctness proof? On the other hand, there is the work of Kohlenbach, who uses classical logic to extract computational content from classical proofs; presumably, implementing his proofs would be done with classical logic.

To tell the truth, these are matters that don’t interest me that much. I’m definitely a mathematician (an analyst), not a logician or a computer scientist!

CC: Give us a simple example of a classical theorem which is not constructively valid? What is the main reason?

DB: A nice example is the intermediate value theorem, which says that if a continuous function $f$ is negative at 0 and positive at 1, then it must take the value 0 somewhere strictly between 0 and 1. Our geometric intuition tells us that this is obvious, as first-time students of analysis are likely to point out. But the theorem does require proof, the usual proof being based on an interval-halving argument: if $f(1/2) = 0$, we are through; if $f(1/2) < 0$, repeat the whole process using the interval $[1/2, 1]$; and if $f(1/2) > 0$, repeat the process using the interval $[0, 1/2]$; and so on. This produces a nested sequence of closed intervals whose intersection is the desired point $c$ with $f(c) = 0$. Unfortunately, this proof, and the theorem itself, is nonconstructive: for at the very first step we may not be able to decide whether $f(1/2)$ equals, is less than, or is greater than 0. We could get away with a decision that $f(1/2) \geq 0$ or $f(1/2) \leq 0$, but even that is not possible constructively. In fact, the intermediate value theorem in its full classical form is constructively equivalent to the statement

$$\forall x \in \mathbb{R} \ (x \geq 0 \lor x \leq 0),$$

where $\mathbb{R}$ denotes the set of real numbers. This statement is false in a recursive interpretation, and is constructively equivalent to the “omniscience principle”

**LLPO:** For each binary sequence $(a_n)_{n \geq 1}$ such that $a_m a_n = 0$ for all distinct $m$ and $n$, either $a_{2n} = 0$ for all $n$ or else $a_{2n+1} = 0$ for all $n$. 

---
The Bulletin of the EATCS

a weak form of the law of excluded middle.

Fortunately, there are versions of the intermediate value theorem that do go through constructively. One is a “same hypotheses, weaker conclusion” version: under the usual hypotheses of the intermediate value theorem, for each $\varepsilon > 0$ there exists (we can construct) $c$ with $0 < c < 1$ and $|f(c)| < \varepsilon$. Another is a “stronger hypotheses, same conclusion” version: if our continuous function $f$ is locally nonzero, in the sense that for each $x \in [0, 1]$ and each $\varepsilon > 0$ there exists $y \in [0, 1]$ such that $|x - y| < \varepsilon$ and $|f(y)| > 0$, then there exists $c$ with $0 < c < 1$ and $f(c) = 0$. The latter version applies to virtually any situation in real analysis that one can think of.

CC: Can you reminiscence about Bishop with whom you co-authored a very important book?

DB: I have to be cautious here, as I only met Bishop twice: once, over coffee, when my wife and I visited San Diego in 1979, and later, at the end of 1982, when I had the privilege of staying with Errett and Jane Bishop in their home in La Jolla while he and I were working on the book. My first actual contact with Bishop (other than reading his book and papers) was while I was a D.Phil. student at Oxford and we engaged in some correspondence over some small things of mine on function algebras. What I wrote must have seemed pretty obvious and trivial to Errett, but he was kind and encouraging in his replies. I believe that kindness and encouragement towards developing mathematicians were a feature of the man.

When I stayed with the Bishops in 1982, I was treated as one of the family, which was very good of them: having an almost-stranger as a house guest is seldom easy. What impressed me most about Errett’s mathematical abilities was the combination of speed and depth: I have never dealt with a mathematician whose deep thought processes were as fast. I mentioned this some years later to Paul Halmos, who had been Errett’s Ph.D. supervisor at Chicago. His response was that he had met one other mathematician who was as fast a thinker as Errett: von Neumann! (In his early career, Halmos had been von Neumann’s research assistant.)

Errett had wide-ranging interests. He was a keen, highly competitive tennis player (wiping me off the court on the one occasion we played). He collected geodes and other rocks. He was widely read in many subjects. Altogether, he was one of the most remarkable intellects and people that I have ever met. It was a great tragedy when he succumbed to cancer in the spring of 1983, a mere four months after I left his home.

CC: Bishop is also famous for the essay “Schizophrenia in contemporary mathematics” (1972). His analysis included the following “principles” (which are relevant for the foundations of constructivism, but seem to go beyond them):
(A) Mathematics is common sense.
(B) Do not ask whether a statement is true until you know what it means.
(C) A proof is any completely convincing argument.
(D) Meaningful distinctions deserve to be preserved.

How relevant are his ideas today? Is mathematics less or more “schizophrenic” than in 1972?

DB: I always thought that “Schizophrenia” was an odd word to use, and was never quite sure what he meant by it. Perhaps he was referring to the mathematicians’ desire to have their cake and eat it, by accepting nonconstructive proofs as adequate for establishing the existence of objects that, at heart, one would really like to “grasp” in some way. I’m reminded of the days when I started a Ph.D. on von Neumann algebras (a Ph.D. I gave up in order to go teaching in high school for a few years). In that subject, a typical argument had the following form. You want to prove the existence of a projection with a certain special property in the von Neumann algebra, so you begin by assuming that no such projection exists; using Zorn’s lemma, you then “construct” a maximal family of projections of some type or other, a type that exists because of your initial assumption; after some pages of careful and deep argument, you arrive at a contradiction, from which you conclude that the projection you originally sought was there after all! These proofs were very clever and beautiful, but left me with a vague sense of unease, which became less vague once I found Bishop’s 1967 book.

Proofs of that sort still abound in mathematics, and it is hard to see how at least some parts of our subject could be handled in any other way: think of the work on higher cardinals. But what we have seen is a substantial increase in awareness of, and interest in, constructive/computable mathematics in the past 20-30 years. It is a lot easier these days than it was when I started giving seminars in the 70s to persuade mathematicians that the distinction between “constructive existence” and “idealised existence” (i.e. existence proved by a contradiction argument) is meaningful. In the earlier days, many excellent mathematicians seemed incapable of appreciating this distinction, I suspect because they had been brought up in the dominant Bourbaki tradition.

I imagine that the improvement in perception of constructive issues has originated in part from mathematicians having computers on their desks and quickly becoming aware that it is not as easy as they though to write programs that work. (Incidentally, Bishop told me that one of the things that made him enter the constructive domain was an unsuccessful attempt he made to picture certain hypersurfaces in complex $n$-space, as part of his famous wok in several complex variables. My recollection is that he had tried to program a machine to draw those surfaces, and had convinced himself that this was an impossible task. In fact, he may have come up with a Brouwerian example showing this; if so, he never made it avail-
The Bulletin of the EATCS

able to others.) Whatever the case, constructive mathematics and its relatives are definitely held in higher, or at least less low, regard in 2009 than they were in 1972.

Anyway, to get back to Bishop’s four principles. Of those, (B) and (D) seem reasonable and ever-applicable, especially (D). Principle (A) is a value judgement, and is perhaps a bit dubious: things may be “common sense” to the super-gifted like Errett, but totally mysterious to us lesser mortals. Principle (C) does not, I think, stand up to philosophical scrutiny. What does it take to be “convincing”? How do I know that I am convinced? And even if I am convinced, how do I know that my conviction coincides with reality? (There are, it is said, still some people who are convinced that the earth is flat.)

CC: Martin-Löf’s constructive type theory is very important in theoretical computer science. Is Martin-Löf’s theory used in constructive mathematics?

DB: Martin-Löf’s type theory is a very important part of constructive mathematics, so much so that (see my article Constructive Mathematics in the Stanford Online Encyclopedia of Philosophy) it can be regarded as a variety of constructive mathematics akin to, but not identical with, that of Bishop. Type theory plays a big part in foundational issues and in the formal (point-free) topology of Sambin and Martin-Löf [10]. However, there are other foundations for constructive mathematics: notably, various set theories such as Myhill’s CST [8], Friedman’s IZF [6], and the Aczel-Rathjen CZF [1] that is currently the most favoured of the three. In my thesis I gave a constructive development of the very formal set theory of [7]; but there are only a couple of things in print about that [3, 4], and it has not been taken further. (It would be a good project for an M.Sc. student to look closely at that work, since the formal language of Morse is more or less a programming one.)

One advantage of Martin-Löf’s type theory over those other foundations for constructive mathematics is that it really is a kind of programming language. I recall years ago seeing a preprint by Jan Smith, which dealt with something like Quicksort in Martin-Löf’s type theory.

CC: How much constructive mathematics should go into the curriculum of mathematics and computer science?

DB: At the undergraduate level, little if any. What I think is important in mathematics at that level is (as I do) to point out the “meaningful distinction” wherever it arises, so that students are well aware that much of the mathematics they see has either little or very obscured computational content. At the honours level it is reasonable to present some constructive analysis in specialised courses—for example, ones on foundations of mathematics.

As regards computer science courses, I’m not really qualified to comment on
how much constructive mathematics would be appropriate, or which course it
would go into. Certainly, in New Zealand it would be almost inconceivable to
generate interest in constructive mathematics among computer science majors—
except perhaps in your university, where you have managed to build up a strong
group of researchers and teachers in theoretical computer science.

CC: Professor Giuseppe Longo made recently an interesting remark: for physics,
non-computability is more important than computability. Is there a similar phe-
nomenon for constructive mathematics and physics?

DB: The relation between constructive mathematics and physics has been the sub-
ject of some controversy, with one philosopher arguing strongly and from several
physical perspectives that constructive mathematics cannot be applied to physics.
I believe, and have written papers to that effect, that this view is based on technical
misunderstanding. But it remains to be seen how, if at all, work like we do in
constructive analysis can be applied to theoretical physics. To date, the one major
piece of work in that area is Richman’s proof (with minor assistance from me) of
Gleason’s theorem [9]; but that theorem, though deep and interesting in its own
right, does not seem to have much real influence on the foundations of quantum
theory. In order to make major applications to modern physics, we would need,
on the one hand, a constructive development of operator-algebra theory (of which
we have very little at present) and, on the other, one of the theory of differential
manifolds (some of which has been given by Diener [5]).

One thing seems clear, though: since, in constructive mathematics, we do not
distinguish between “computable” and “non-computable” objects—if an object
is well defined constructively, it has to be computable—non-computability will
not be a part of Bishop-style constructive physics. Of course, we may produce
Brouwerian examples showing that certain classical theorems cannot be proved
constructively, and we may even go so far as to work in the recursive model of
Bishop’s mathematics, in which explicit examples of non-computable things can
appear; but as long as we work within Bishop’s framework unadorned with such
extras as the Church-Markov-Turing thesis, then we will be producing only posi-
tive results, such as (perhaps) a constructive proof of the existence of a black hole
under certain hypotheses.

CC: Please comment on the relevance of constructive mathematics for economical
theories.

DB: In view of the allegations made about the role of mathematical models in
the recent global economic collapse, one might ask what is the relevance of any
mathematics for economic theory!

Now, as you are aware, I have indulged in some constructive aspects of eco-
nomic theory, such as the existence of preference relations and demand functions.
While these investigations were extremely interesting to me, and in some cases spawned non-constructive work by me and others, it would be ridiculous for me to claim that their outcome had any real relevance to economics. I suspect that that holds for much of the classical work going on in mathematical economics: it generates beautiful, hard mathematics based on, but very far removed from, real-world economic problems.

That is not to say that one shouldn’t continue looking constructively at problems that arise from an economic situation. We just need to be realistic about the non-mathematical merits of our solutions to them.

CC: Tell us about your extra-mathematical interests?

DB: Until my knee succumbed to years of running, I was an active runner and cricketer, playing for the Staff XI at Waikato University during my tenure there. At that time I also coached junior football (not rugby, but the real game, with the round ball), starting with my elder son’s team of 12-year-olds and going right through his high-school days. Occasionally I play what may look a bit like golf but is a vague shadow of the game I played as a boy and young man before leaving Scotland. I still cycle, though that has lapsed a bit in the past year.

These days my major interest is classical music, and in particular my choir, Musica Balkanica, which was founded in Christchurch by some expatriate Serbs and is devoted to singing music from the Balkan countries. How I got into that choir is another story, but the experience of singing in many languages unknown to me, and of sharing time and friendship with my fellow singers, has been great; it has also fostered my long-standing interest in language and languages. I read a great deal, with particular interest in biographies. And once a month I am Service Leader at St Christopher’s Church.

CC: Many thanks.

References


The EATCS Columns
The problem of Identity Testing consists in given an arithmetic circuit that computes a polynomial $p$ in a field, decide whether $p$ is the zero polynomial. One of the first examples of probabilistic algorithms is the polynomial time randomized solution to this problem given by Schwartz and Zippel. More recently there has been considerable progress in trying to find a polynomial time deterministic solution to this important problem at the borderline between complexity theory and algebra.

Nitin Saxena, one of the experts in the area, gives in this survey a beautiful overview of several recent results dealing with the complexity of Polynomial Identity Testing.

Progress on Polynomial Identity Testing

Nitin Saxena
Hausdorff Center for Mathematics,
Endenicher Allee 62, 53115 Bonn, Germany, ns@hcm.uni-bonn.de

Abstract

Polynomial identity testing (PIT) is the problem of checking whether a given arithmetic circuit is the zero circuit. PIT ranks as
one of the most important open problems in the intersection of algebra and computational complexity. In the last few years, there has been an impressive progress on this problem but a complete solution might take a while. In this article we give a soft survey exhibiting the ideas that have been useful.

1 Introduction

One learns a number of identities as part of Algebra in the school curriculum. For example, the difference of squares identity \((x+y)(x-y) = (x^2 - y^2)\) or a more impressive sum of four squares identity (probably first communicated by Euler in a letter to Goldbach on May 4, 1748): 

\[
(a_1^2 + a_2^2 + a_3^2 + a_4^2)(b_1^2 + b_2^2 + b_3^2 + b_4^2) = (a_1b_1 - a_2b_2 - a_3b_3 - a_4b_4)^2 + (a_1b_2 + a_2b_1 + a_3b_4 - a_4b_3)^2 + (a_1b_3 - a_2b_4 + a_3b_1 + a_4b_2)^2 + (a_1b_4 + a_2b_3 - a_3b_2 + a_4b_1)^2.
\]

There is of course an easy way to test them: just completely expand the products and check whether the monomials cancel in the resulting sum. This way you can easily verify that the above two expressions are indeed identities. But the process of expanding products blows up monomials and would be very expensive when both the number of variables and the degree of the expression are increased. Roughly, for \(n\) variables and \(d\) degree the number of monomials grows as \(\binom{n+d}{d}\) which is small if one of \(n\) or \(d\) is small but exponentially large if both \(n, d\) are large. So the question we ask is - can this identity or zero testing be done in \((nd)^{O(1)}\) steps?

Notice that to formalize this question there seems to be a need of carefully defining the way the algebraic expression is given to us in the input. Fortunately, there is already an object defined in computational complexity, called arithmetic circuit, that we could directly use [48]. An arithmetic circuit \(C\), on say \(n\) variables and a field \(\mathbb{F}\), is a directed acyclic graph with input variables at the leaves and output at the root. The internal nodes are called gates, they are of two kinds - multiplication and addition - and perform the respective operations over the field \(\mathbb{F}\). The edges or wires of \(C\) can have constants on them from the field which get multiplied to the value at the tail of the respective edge. It is easy to see that the value at the root of the circuit \(C\) is just an \(n\)-variate polynomial and lives in \(\mathbb{F}[x_1, \ldots, x_n]\). Thus a circuit is a natural combinatorial way to capture algebraic computation. One might want to consider the base object \(\mathbb{F}\) to be a general ring instead of a field but we will be more concerned with the latter in this survey. For the purposes of identity testing we usually regard the operations in the field \(\mathbb{F}\) to be doable in unit time. The bulk of the computation in the identity testing algorithms is seen as a function of the size of the input circuit, which is basically the...
number of gates and wires in the circuit. Another useful parameter of a circuit is depth, which is the number of levels between the root and the leaves. Fanin/Fanout refers to the maximum number of inputs/outputs a gate has in the circuit, and a circuit with fanout 1 is called a formula. Finally, we use the notation $\text{poly}(s, t)$ to denote a positive-valued function whose asymptotic behaviour is $(s + t)^{O(1)}$. The problem of identity testing is then:

**Problem 1.1 (PIT).** Given an arithmetic circuit $C$ in the input that computes a polynomial $p(x_1, \ldots, x_n)$ in $\mathbb{F}[x_1, \ldots, x_n]$. Find a deterministic algorithm that tests if $p$ is the zero polynomial, and uses only $\text{poly}(\text{size}(C))$ many $\mathbb{F}$ operations.

PIT is currently an open question and, as we will see in this survey, an important question in complexity theory. But it has an easy “practical” solution, i.e. there are randomized polynomial-time algorithms that are easy to implement. The first randomized polynomial time algorithm was given (independently) by Schwartz [39] and Zippel [49]. It simply evaluates the input circuit at a randomly chosen point in $\mathbb{F}^n$ and outputs YES iff the specific evaluation is zero. This idea works mainly because a nonzero polynomial cannot have “too many” roots over a field:

**Lemma 1.2 (Schwartz-Zippel).** Let $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a non-zero polynomial of degree $d \geq 0$ over a field $\mathbb{F}$. Let $S$ be a finite subset of $\mathbb{F}$. Then,

$$\text{Prob}_{r_1, \ldots, r_n \in S}[P(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}.$$ 

**Proof.** The proof is by induction on $n$. For $n = 1$, $P$ can have at most $d$ roots and hence the probability of hitting a root is at most $\frac{d}{|S|}$.

Now, assume that the statement holds for all polynomials upto $(n - 1)$ variables. Wlog we can then consider $P$ to be a polynomial in $x_1$ by writing it as,

$$P(x_1, \ldots, x_n) = \sum_{i=0}^{d} x_1^i P_i(x_2, \ldots, x_n).$$

Since $P$ is a nonzero polynomial, $\exists i$ such that $P_i$ is nonzero. Take the largest such $i$, clearly $\text{deg} P_i \leq (d - i)$. Now we randomly pick $r_2, \ldots, r_n$ from $S$. By the induction hypothesis, $\text{Prob}[P_i(r_2, \ldots, r_n) = 0] \leq \frac{d - i}{|S|}$. If $P_i(r_2, \ldots, r_n) \neq 0$ then $P(x_1, r_2, \ldots, r_n)$ is of degree $i$ so by the univariate case:

$$\text{Prob}[P(r_1, \ldots, r_n) = 0 | P_i(r_2, \ldots, r_n) \neq 0] \leq \frac{i}{|S|}.$$
By a lazy probability estimation we get:

\[
\Pr_{r_1, \ldots, r_n \in S} [P(r_1, \ldots, r_n) = 0] \leq \Pr_P [P(r_2, \ldots, r_n) = 0] + \\
\Pr [P(r_1, \ldots, r_n) = 0 \mid P_i(r_2, \ldots, r_n) \neq 0] \leq \frac{d - i}{|S|} + \frac{i}{|S|}
\]

\[
\leq \frac{d}{|S|}
\]

Thus completing the proof by induction. \(\square\)

This lemma shows that as long as the field \(F\) has twice as many elements as the degree of the input circuit, we have a good randomized algorithm that has error probability at most \(\frac{1}{2}\). In case \(F\) is too small compared to the degree \(d\) of the input circuit \(C\) we go to a suitable extension of \(F\) and pick random points there. The two issues here that are worth mentioning: (1) The degree of the input polynomial can only be at most \(2^{\text{size}(C)}\). (2) A field extension of \(F\) of degree \(O(\text{size}(C))\) can either be found by a deterministic construction of irreducible polynomials [8] or we can simply work over the cyclotomic extension \(\mathbb{F}[z]/(z^r - 1)\) for a “suitable” \(r = \text{poly}(\text{size}(C))\). Finally, the actual evaluation of \(C\) at a randomly chosen point (even from the extension algebra) can be trivially simulated in \(\text{poly}(\text{size}(C))\) many \(F\) operations.

Randomized algorithms that use fewer random bits and have lower error probability (at the cost of time) were given by Chen & Kao [17], Lewin & Vaidhan [35], and Agrawal & Biswas [1]. As we care more about deterministic methods in this survey, we will not discuss the details of these methods here. These randomized algorithms show that PIT is in the complexity class BPP which in turn is conjectured to be equal to \(P\) (see the survey [27] for the theoretical evidence). Thus it seems to be a reasonable goal to derandomize PIT.

Some applications of PIT

Being a fundamental problem it is not surprising that PIT appears in several other seemingly unrelated problems. We see below an example each from complexity theory, graph theory and number theory.

The idea of comparing two multivariate polynomials for equality by evaluating them at randomly chosen points was crucial in the proof of the complexity result: \(\text{IP} = \text{PSPACE}\) [41]. The multivariate polynomial in that case is the arithmetized version of a quantified boolean formula (QBF) \(\phi\), and using PIT it becomes possible to give an interactive protocol (IP) to verify
the truth of $\phi$. Here PIT helped in upper bounding the complexity of QBF problem, on the other hand, PIT also has several lower bound implications (see Section 7)

Another, much older, application of PIT is due to the following theorem proved by Tutte [47] in 1947: A graph has no perfect matching iff the determinant of its Tutte matrix is zero. Recall that for a graph $G = (V, E)$ on $n$ vertices its Tutte matrix is an $n \times n$ matrix $A$ with its $(i, j)$th entry defined as:

$$A_{i,j} := \begin{cases} x_{i,j}, & \text{if } (i, j) \in E \text{ and } i < j \\ -x_{j,i}, & \text{if } (i, j) \in E \text{ and } i > j \\ 0, & \text{otherwise} \end{cases}$$

To use this theorem in a matching algorithm we will have to check whether the multivariate polynomial $\det(A)$ is zero, which can be seen as a special case of PIT. This formulation immediately gives a randomized algorithm which has the added advantage of being in (randomized) NC, i.e. it is a highly parallel algorithm since determinant has known fast parallel algorithms. It is an open question to find a deterministic parallel algorithm for perfect matching, and it appears that a derandomization of this special case of PIT might be the way to go (see a related conjecture in [2] and a special case in [5]).

Finally, the problem of primality testing was solved in an elementary way by working with a PIT formulation. It was observed by Agrawal & Biswas [1] that a positive integer $n$ is prime iff $(x + 1)^n = (x^n + 1) \mod n$, and they exploited this simple binomial fact to design a new randomized primality test. If we define $P(x) := (x + 1)^n - (x^n + 1)$ then the question is that of testing whether $P(x)$ is the zero polynomial over the ring $\mathbb{Z}/n\mathbb{Z}$, which is just a special case of PIT. Note that although $P(x)$ is a univariate polynomial it has degree $n$ which is exponential in the input size $\log n$, and so we cannot afford to completely expand $P(x)$. The neat idea in [1] was to test $P(x) = 0 \mod n, Q(x))$ for a randomly chosen polynomial $Q$ of degree $O(\log n)$. As $Q$ has “small” degree we can do this in $\text{poly}(\log n)$ time, using repeated squaring of $(x + 1)$ and $x$. This randomized algorithm was later derandomized by Agrawal, Kayal & Saxena [7] to get the first deterministic polynomial time algorithm for primality testing. They essentially showed that if $P(x) = 0 \mod n, a^r x^r - 1$ for all $1 \leq a, r \leq (\log n)^2$ then $P(x) = 0 \mod n$. It is astonishing that the zeroness of a polynomial of a high degree can be determined by just looking modulo very few, very small polynomials!
Survey Overview

The goal of this survey is not to be exhaustive but to cover the main ideas and to pose the closely related open questions. One of the interesting topics related to PIT which we would not be discussing in this survey are: PIT for circuits over general rings (see [44]), interpolation of polynomials (see [15, 23, 42]) and learning arithmetic formulas (see [45, 33]). A brief overview of the topics that we do cover in this survey now follows.

Sparse PIT. A circuit $C$ that computes a polynomial which has at most $m$ nonzero monomials is called $m$-sparse. The problem of sparse PIT is to design an algorithm that runs in $\text{poly}(\text{size}(C), m)$ field operations. This problem has several known solutions (see [30]). We will see the one by Agrawal [3] as I find it the simplest conceptually.

Low Degree PIT. A circuit $C(x_1, \ldots, x_n)$ that computes a polynomial of degree $\text{poly}(n)$ is called a low degree circuit. The term low degree is used to contrast with circuits that use repeated squaring to exponentially increase the degree, for example the circuit $P(x)$ that appears in the primality test above is not low degree. It can be seen that formulas, circuits of constant depth, and bounded fanin circuits of $O(\text{log } n)$ depth; all compute a low degree polynomial. Thus, it seems natural to study the problem of PIT for low degree circuits and we call it low degree PIT.

It was shown by Agrawal & Vinay [10] that for the purposes of low degree PIT it is enough, somewhat surprisingly, to just consider depth-4 circuits. The main idea is to “shrink” any low degree circuit into a depth-4 circuit by paying only a subexponential price in the circuit size. Thus if one solves depth-4 PIT in deterministic polynomial time then one has solved low degree PIT in subexponential time. I mainly see this as a strong evidence that PIT for “shallow” circuits, i.e. those of depth 3 or 4, gives us enough clues to tackle the bigger PIT problem.

Depth-3 PIT (Non Black-Box). Convinced that shallow circuits are already interesting cases for PIT, we now focus on the PIT algorithms for depths 2, 3 and 4. A depth 2 circuit is either a sum of monomials ($\Sigma \Pi$) or a product of linear polynomials ($\Pi \Sigma$), both of which have obvious deterministic polynomial time PIT algorithms if we can look “inside” the circuit (which is why we use the term non black-box). A depth 3 circuit can either be a product of sum of monomials or a sum of product of linear polynomials. As the former case is again trivial to check for zero-ness, we only worry about the latter case. Thus, for us a depth-3 circuit $C$ over a field $F$ is $C(x_1, \ldots, x_n) = \sum_{i=1}^{d} T_i$, where $T_i$ (a multiplication term) is a product of $d_i$ linear polynomials $L_{i,j}$ over $F$. Note that by homogenization we can assume wlog that $L_{i,j}$’s are linear forms (i.e. linear polynomials with a zero constant coefficient) and that
The Bulletin of the EATCS

\[ d_1 = \cdots = d_k =: d. \]  Such a circuit is referred to as a \( \Sigma\Pi\Sigma(n, k, d) \) circuit, where \( k \) is the top fanin of \( C \) and \( d \) is the degree of \( C \). Depth-3 circuits are a good starting point and are under intense study from various viewpoints [22, 19, 31, 9, 45, 32, 38, 42, 33, 34, 43].

It was shown by Kayal & Saxena [31] that if the top fanin \( k \) is small then PIT is easy. Note that \( k = 2 \) is the trivial case (because \( \mathbb{F}[x_1, \ldots, x_n] \) is a unique factorization domain) but \( k = 3 \) is already nontrivial. The main idea of [31] was to look at \( C \) modulo several linear forms and use a generalized form of Chinese remaindering. The cost of doing this grows like \( d^k \) and hence is meaningful when \( k \) is small even if \( d, n \) are arbitrarily large. As one needs to look “inside” the circuit this algorithm we label as non black-box.

**Depth-3 PIT (Black-Box).** One might wish to develop algorithms for PIT that do not even look inside a given circuit \( C \), but merely evaluate \( C \) at several points in \( \mathbb{F} \) or algebraic extensions of \( \mathbb{F} \). Of course this is an impossible dream if we do not have an \textit{a priori} bound on \( \text{size}(C) \). But with such a bound given in the input together with a black-box access to \( C \), the question of testing \( C = 0 \) in \( \text{poly}(\text{size}(C)) \) many \( \mathbb{F} \) operations becomes reasonable, as the randomized Schwartz-Zippel PIT algorithm does not look inside the circuit at all! Intuitively it seems that to devise a black-box PIT algorithm for a circuit family, one would need a very good understanding about the structure of identities in that family. There is some progress in that direction for \( \Sigma\Pi\Sigma(n, k, d) \) identities with constant top fanin \( k \).

Note that a \( \Sigma\Pi\Sigma(n, k, d) \) identity \( C \) is composed of linear forms and hence we can associate a natural notion of \textit{rank}, which will be the rank of the vector space that these linear forms span. It was first shown by Dvir & Shpilka [19] that, under some mild assumptions on \( C \), the rank of \( C \) is bounded by \( \log^k d \). For a constant \( k \) this is saying something nontrivial about the \( \Sigma\Pi\Sigma(n, k, d) \) identities. Later Karnin & Shpilka [32] used this property to develop a black-box PIT algorithm for \( \Sigma\Pi\Sigma(n, k, d) \) circuits that runs in time \( \sim d^{\text{rank}(C)} \), or \( d^{\log^k d} \) which is a subexponential complexity when \( k \) is constant. This connection between rank and black-box PIT is quite encouraging, and has already led to several improvements. Saxena & Seshadhri [43] showed a rank bound of \( k^3 \log d \) which is almost optimal and translates into an improved black-box PIT algorithm of complexity \( d^{k^3 \log d} \). Their main idea was to look at \( C \) modulo various ideals and deduce lots of dependencies between the various multiplication terms of the identity \( C \).

It is believed that over the fields of zero characteristic, especially complex numbers, the identities should be even more restricted. Towards that goal, Kayal & Saraf [34] showed a rank bound of \( k^k \) over the field of \textit{reals}. It gives a corresponding black-box PIT algorithm of complexity \( d^{k^k} \), which is
polynomial time for constant $k$. Their main idea is to look at the linear forms appearing in $C$ as points in a higher dimensional space and then use certain properties of real geometry [13] to rule out their arrangement in a $\Sigma \Pi \Sigma(n, k, d)$ identity.

**Depth-4 PIT.** PIT algorithms for circuits of depth higher than 3 are currently few [37, 9, 38, 45, 46]. The ones that are known are based on insights obtained from depth-2 and depth-3 circuits and put further restrictions so that these ideas could be lifted to higher depths. We will discuss the PIT algorithms for noncommutative formulas [37] and depth-4 circuits with multiplication gates that only do powering [38].

A noncommutative formula is one that has noncommuting variables, i.e. $x_i x_j \neq x_j x_i$ for all $i \neq j \in [n]$. The main idea of Raz & Shpilka [37] was that a multiplication gate in a noncommutative formula can be gradually opened-up without getting into the problem of monomial explosion. They then used linear algebra to complete the PIT algorithm.

Saxena [38] solved the case of depth-4 circuits when each multiplication gate is just powering, i.e. an input $p(x_1, \ldots, x_n)$ is converted to $\alpha \cdot p(x_1, \ldots, x_n)^e$ for some $\alpha \in \mathbb{F}$ and $e \in \mathbb{N}$. The main idea was to transform such a circuit to another one wherein each multiplication gate has factors with unmixed variables. The PIT algorithm then follows an algebraic generalization of the idea of [37].

**General PIT and Lower Bounds.** As seen above PIT is a fascinating fundamental problem with direct connections to other problems. As if this was not enough, Kabanets & Impagliazzo [29] further emphasized the importance of PIT by showing that a complete solution of PIT would imply circuit lower bounds. They showed that if PIT is in P then either Permanent (the naughtier sibling of Determinant) does not have polynomial sized arithmetic circuits or NEXP does not have polynomial sized boolean circuits. Even though we “believe” both the conclusions to be independently true, nevertheless, the connection with PIT is intriguing. Their main idea is to show that low-degree-PIT$\in$P together with the existence of small circuits for permanent and those for NEXP implies NEXP$\subseteq$NP, which is a contradiction. In the proof PIT is only used to test whether a small arithmetic circuit equals the permanent function.

In a more explicit way, Agrawal [3] showed that if there are black-box PIT algorithms for a circuit family then they also exhibit lower bounds for that family.
Sparse PIT

A lot of papers have focused on the case of circuits that compute a sparse polynomial. In this case we are given a circuit $C$ together with an upper bound $m$ on the number of nonzero monomials in the computed polynomial, and the goal is to devise a $\text{poly}(\text{size}(C), m)$ time PIT algorithm. Notice that this is a "benign" goal as usually in PIT a given circuit would produce an exponential (in $\text{size}(C)$) number of nonzero monomials.

There exist a host of solutions for this case and also for the seemingly more general problem of interpolating such circuits [15, 23, 16, 30, 3, 5, 12]. All these algorithms are based on the idea of evaluating the given circuit at cleverly chosen points so that a specific nonzero monomial gets isolated. Since there are few nonzero monomials one of them can be efficiently isolated by just doing evaluations, hence these tend to be black-box algorithms. We exhibit one such algorithm, following Agrawal [3]. The basic idea is to go to a cyclotomic extension and evaluate the circuit at the virtual roots of unity available in this extension algebra.

**Theorem 2.1.** Let $p(x_1, \ldots, x_n)$ be a nonzero polynomial (over a field $\mathbb{F}$) whose degree in each variable is less than $d$ and the number of monomials is at most $m$. Then there exists an $1 \leq r \leq (mn \lg d)^2$ such that, $p(y, y^d, \ldots, y^{d^{n-1}}) \neq 0 \pmod{y^r - 1}$.

**Proof.** Consider the polynomial $q(y) := p(y, y^d, \ldots, y^{d^{n-1}})$ in $\mathbb{F}[y]$. Note that a monomial $x_1^{i_1} \cdots x_n^{i_n}$ in $p$ is mapped to the monomial $y_1^{i_1+d} \cdots y_n^{i_n+d^{n-1}}$ in $q$. Since we have assumed $i_1, \ldots, i_n < d$, observe that the map is one-to-one. Consequently, $q(y) \neq 0$. Say $y^b$ is a monomial with nonzero coefficient in $q$. Now we look at $q(y)$ modulo $(y^r - 1)$.

If $q(y) = 0 \pmod{y^r - 1}$, then there ought to be another monomial $y^b \neq y^a$ with nonzero coefficients in $q(y)$ such that $y^b \equiv y^a \pmod{y^r - 1}$. This is possible iff $r| (b - a)$. Thus, to avoid picking such a "bad" $r$ we need one that satisfies:

$$r \nmid \prod_{y^b \in q(y), b \neq a} (b-a) =: R.$$

Clearly, integer $R$ can be at most $(d^n)^m$ in value. Since $R$ has at most $\lg R$ prime factors and since we would encounter at least $(\lg R)^2 = (mn \lg d)^2$, it is clear that we have the required (prime) $r$ for which $q(y) \neq 0 \pmod{y^r - 1}$.

**Algorithm.** The above property immediately gives a black-box PIT algorithm for sparse polynomials. Given an $m$-sparse circuit $C(x_1, \ldots, x_n)$ over

2 Sparse PIT
\( \mathbb{F} \), fix \( d := 2^{\text{size}(C)} \) and for every \( 1 \leq r \leq (mn \lg d)^2 \): compute \( d, d^2, \ldots, d^{n-1} \) modulo \( r \) using repeated squaring and then evaluate \( C(y, y^d, \ldots, y^{d^{n-1}}) \) over the extension algebra \( \mathbb{F}[y]/(y^r - 1) \). Finally, we declare \( C \) to be an identity if all these evaluations are zero. It is routine to verify that this is a correct algorithm with time complexity \( \text{poly(size}(C), m) \).

**Open.** One might wonder what happens if we replace the parameter \((mn \lg d)^2\) in the above analysis by a milder parameter like \( \text{poly(size}(C)) \)? If we could still prove the statement in Theorem 2.1 then we would get a conceptually simple black-box algorithm for general PIT! Such a generalization of Theorem 2.1 is currently an open question, even for the “smallest” case of depth-3 circuits. (Note that the theorem trivially applies to the case of depth-2 circuits.) It is conjectured by Agrawal [3] that Theorem 2.1 should be true if we replace \((mn \lg d)^2\) by \( \text{size}(C) \text{depth}(C) \), thus, solving PIT at least for constant depth circuits.

### 3 Low Degree PIT

The circuit model is a very expressive representation for polynomials, for example, in size \( s \) it is possible to achieve degree \( 2^s \) by repeated squaring (although the number of monomials produced remains singly-exponential in \( s \) and not doubly-exponential). What if we reduce the expressive nature of a circuit, say, by restricting the degree of the computed polynomial to be “only” \( \text{poly}(s) \)? Intuitively, PIT for these circuits should be easier.

It was shown by Agrawal & Vinay [10] that a low degree circuit \( C(x_1, \ldots, x_n) \), i.e. of degree \( \text{poly}(n) \), can be shrunk to a depth-4 circuit by a reasonable blowup in the size. We will now see the main idea of the proof. As we can always add some useless variables to \( C \), we can assume wlog that \( C(x_1, \ldots, x_n) \) is computing a polynomial of degree \( d = O(n) \). Furthermore, as any \( n \) variate, \( d \) degree polynomial can be trivially computed by a depth-2 circuit of size \( \sim (n + d)^d = 2^{\text{poly}(\log d)} \), it is only reasonable to assume that \( C \) has size \( 2^{\text{poly}(\log d)} \) (note the small \( o \) in the exponent). In that case the theorem of [10] states:

**Theorem 3.1.** If a polynomial \( P(x_1, \ldots, x_n) \) of degree \( d = O(n) \) has a circuit \( C \) of size \( 2^{\text{poly}(\log d)} \) then there is a depth-4 circuit \( C' \) of size \( 2^{\text{poly}(\log d)} \). Moreover, it can be explicitly constructed in \( 2^{\text{poly}(\log d)} \) time, given \( C \) in the input.

**Proof Sketch:** The depth reduction is done in two stages. The first stage reduces the depth to \( O(\lg d) \) by an efficient construction of Allender, Jiao, Mahajan and Vinay [6]. The second stage is more expensive but it reduces the depth to 4.
The Bulletin of the EATCS

The main idea in the first stage is to look at certain intermediate polynomials \([g, h]\) computed inside the circuit \(C\): for any gate \(g\) in \(C\) and any gate \(h\) in the subtree rooted at \(g\), \([g, h]\) is defined to be the polynomial computed at the node \(g\) if the subtree at \(h\) is replaced by a leaf labelled 1. This immediately gives us the simple relation: \(C(x_1, \ldots, x_n) = \sum_{\text{root}(C)} x_r x_s\). Next the polynomial \([g, h]\) is recursively expanded wrt the multiplication gates \(p\) in the subtree (rooted at \(g\)) for which the degree \(\geq \frac{1}{2}\) \(\deg(gh)\) > degree of the children of \(p\). This expansion is then used to construct a circuit \(C''\) whose gates correspond to \([g, h]\), for “all” gates \(g, h\) in \(C\). A clever argument in [6] shows that every multiplication gate in \(C''\) has at least a doubling effect on the degree of its children, hence, the depth of \(C''\) can be at most \(O(\lg d)\). Also, \(\text{size}(C'')\) remains at most a polynomial in \(\text{size}(C)\).

Let \(s\) be the size of \(C''\) and define a parameter \(\ell\) sufficiently smaller than \(\frac{d \lg \frac{n}{s}}{\lg s}\). The second stage has an even simpler idea: cut \(C''\) into two parts, the top has exactly \(t := \lg \ell\) layers of multiplication gates and the rest of the layers form the bottom. Let \(g_1, \ldots, g_k\) (where \(k \leq s\)) be the output gates of the bottom part. Thus, we can think of the top part as computing a polynomial \(P_{\text{top}}\) in new variables \(y_1, \ldots, y_k\) and each of the \(g_i\) computing a polynomial \(P_i\) in the input variables \(x_1, \ldots, x_n\). The polynomial computed by the circuit \(C''\) then equals: \(P_{\text{top}}(P_1(x_1, \ldots, x_n), \ldots, P_k(x_1, \ldots, x_n))\). Since the top half consists of \(\ell\) levels of multiplication gates \(\deg(P_{\text{top}})\) is bounded by \(2^\ell\). And since the degree drops by a factor of two across multiplication gates, we also have \(\deg(P_i) \leq \frac{d}{2}\). Expressing \(P_{\text{top}}\) and \(P_i\)'s as a sum of product, we have a depth-4 circuit \(C'\) computing the same polynomial as \(C''\). The size of this circuit \(C'\) is:

\[
\sim \left( \frac{k + 2^\ell}{k} \right) + k \cdot \left( \frac{n + \frac{d}{2}}{n} \right)
\]

An easy calculation shows that the dominating terms above are: \(s' + (nt)\frac{d}{2}\). Both of which, by the choice of \(\ell\), are smaller than \(2^{d \lg \frac{n}{s'}}\). This completes the proof.

This theorem already suggests that a PIT algorithm for depth-4 circuits would imply a nontrivial one for low degree circuits. [10] goes a step further and shows that a black-box polynomial time algorithm for depth-4 PIT gives an \(n^{\log n}\) time algorithm for low degree PIT!

Open. The above proof has an interesting byproduct: if we could prove an exponential lower bound for a low degree polynomial for depth-4 circuits then it implies an exponential lower bound for general circuits! For example, we know that permanent (on \(n \times n\) matrices) has a depth-4 circuit of size \(2^{O(n)}\). But whether it has depth-4 circuits of size \(2^{\Theta(n)}\) is not known. Such a lower
bound would now imply that permanent does not have (general) arithmetic circuits of size $2^{o(\sqrt{n})}$.

4 Depth-3 PIT (Non Black-Box)

The case of depth-2 being too easy (it has a black-box polynomial time PIT algorithm) and that of depth-1 being too general (its PIT algorithm will also give a nontrivial one for low degree circuits), leaves us with the intermediate case of depth-3 PIT. There are a host of results for it but the case is still not completely solved. Here we will sketch the idea of the best known PIT algorithm [31]. It is a non black-box algorithm as it needs to look at the input circuit to use the linear polynomials that occur in it.

Let the input circuit $C$ be computing over a field $\mathbb{F}$. As discussed before we can assume wlog that $C$ looks like: $C(x_1, \ldots, x_n) = T_1 + \cdots + T_k$, where $T_i$ is a product of linear polynomials $L_{i,1}, \ldots, L_{i,d}$ i.e. each $L_{i,j} = (a_{i,j,0} + a_{i,j,1}x_1 + \cdots + a_{i,j,n}x_n)$ for some constant $a$'s from $\mathbb{F}$. Note that the case of $k = 2$ is trivial as checking $T_1 + T_2 = 0$ entails comparing the linear factors of $T_1$ and $T_2$, which we know explicitly. Thus, $k = 3$ is the first nontrivial case and indeed therein lies the main idea of [31]. So we sketch the algorithm only for the case $C = T_1 + T_2 + T_3$.

Chinese Remaindering. The starting idea is to study $C$ modulo linear polynomials. So pick $(d+1)$ coprime linear polynomials $p_1, \ldots, p_{d+1}$ from the set $\{L_{i,j} | i \in [3], j \in [d]\}$. Note that by elementary algebra, $C = 0$ iff for all $i \in [d+1]$, $C = 0 \pmod{p_i}$. The latter conditions are easy to check because $C$ modulo $p_i$ is just a sum of two multiplication gates, say $C = T_1 + T_2 \pmod{p_i}$. Now we can further simplify the situation by mapping $p_i \mapsto x_i$ by applying a suitable invertible linear transformation $\tau$ on $x_1, \ldots, x_n$ (i.e. it replaces $x_i$ by a linear combination of the $x$'s). It is easy to see that $C = 0 \pmod{p_i}$ iff $C(\tau(x_1), \ldots, \tau(x_n)) = 0 \pmod{x_i}$. The latter can be tested by simply comparing the linear factors of $\tau(T_1)$ and $\tau(T_2)$ after fixing $x_1 = 0$ in them.

Thus, zero testing of $C$ just boils down to picking the right set of linear polynomials amongst the ones that appear in the definition of $C$. But the above idea fails if the set $\{L_{i,j} | i \in [3], j \in [d]\}$ does not have $(d+1)$ coprime linear polynomials. This could easily happen, for example when $C = x_1^9 + x_2^9x_3^9 - (x_1 + x_2 + x_3)^9$ we have only 4 coprime linear polynomials while we need 10. In that case we require more algebra:

Chinese Remaindering over Ideals. The idea that finally works is to study $C$ modulo “nice looking” ideals. Formally, pick coprime linear polynomials $p_1, \ldots, p_d$ from the set $\{L_{i,j} | i \in [3], j \in [d]\}$ such that there exist
The Bulletin of the EATCS

exponents $e_1, \ldots, e_\ell$ satisfying:

1) every $p_i^{e_i}$ divides some $T_j$.
2) $e_1 + \cdots + e_\ell > d$.

A simple calculation shows that such powers of linear polynomials $p_1^{e_1}, \ldots, p_\ell^{e_\ell}$ always exist (unless the multiplication terms in $C$ are not distinct). Also $C = 0$ iff for all $i \in [\ell]$, $C = 0 \pmod{p_i^{e_i}}$. But how do we check the latter condition? We can again first simplify it to $p_i \mapsto x_1$ by applying an invertible linear map $\tau$ on $x_1, \ldots, x_n$. Then $C = 0 \pmod{x_i^{e_i}}$ iff $C(\tau(x_1), \ldots, \tau(x_n))$ vanishes over the algebra $F[x_1]/(x_i^{e_i})$. Since $\tau(C) \pmod{x_i^{e_i}}$ is a sum of two multiplication gates, say $\tau(T_1) + \tau(T_2)$, testing $\tau(T_1) + \tau(T_2) = 0$ modulo the ideal $(x_i^{e_i})$ boils down to a more sophisticated comparison of the linear factors of $\tau(T_1)$ and $\tau(T_2)$.

**Algorithm & Complexity.** The final PIT algorithm for a $\Sigma\Pi\Sigma(n, k, d)$ circuit $C = T_1 + \cdots + T_k$ thus identifies a set $I$ of nice looking ideals, namely,

$I := \{(f_1, \ldots, f_\ell) \mid \ell \in [k-1], \forall i \in [\ell], f_i$ is a maximal factor of some $T_j$ s.t. $f_i$ is not a zero-divisor modulo $(0, f_1, \ldots, f_{i-1})$ and $f_i$ is power of a linear polynomial modulo the radical of $(0, f_1, \ldots, f_{i-1})\}.$

Actually, the ideals useful for the algorithm of [31] are a subset of $I$ but this set captures the most important properties of the ideals. The PIT algorithm just checks $C = 0 \pmod{I}$, for every $I \in I$, and that implies the zeroeness of $C$. The test $C = 0 \pmod{I}$ is easy to do because $C(\pmod{I})$ is basically just one multiplication gate. Moreover, since $|I| < d^k$ and the dimension of the factor-algebra of any ideal in $I$ is also at most $d^k$, the algorithm has time complexity $\text{poly}(n, d^k)$.

5 Depth-3 PIT (Black-Box)

The depth-3 PIT algorithm seen above is inherently non black-box as it uses the linear polynomials that define the input circuit. It is more desirable to have a black-box PIT algorithm as it will imply circuit lower bounds (see [3, 10]) and it tends to be useful in learning algorithms [23, 33]. Note that the randomized PIT algorithm based on Schwartz-Zippel lemma is black-box, hence, by the conditional derandomizations of BPP [27] such a deterministic polynomial-time black-box PIT algorithm is conjectured to exist.

The black-box version of depth-3 PIT is the one in which we are given a black-box $C(x_1, \ldots, x_n)$ with the promise that $C$ computes a depth-3 circuit.
over a known field $\mathbb{F}$ and has a known size bound $\text{size}(C)$. This question has received a fair amount of attention [32, 43, 34, 46] but is not yet completely solved. The known black-box methods are successful, to a varying degree, only in the case of $\Sigma\Pi\Sigma(n, k, d)$ circuits when the top fanin $k$ is “small”. We discuss in this survey techniques that are based on the notion of rank of a depth-$3$ circuit, i.e. the dimension of the vector space spanned by the linear polynomials that appear in its multiplication terms. For a $\Sigma\Pi\Sigma(n, k, d)$ identity a trivial bound on the rank is $kd$. It is not immediately clear whether the rank of an identity should be significantly smaller than $kd$, but it is and this property helps in developing the black-box PIT algorithms. To show the rank bounds we need to put certain “mild” conditions on the circuits:

**Definition 5.1.** (Minimal and Simple circuits) A $\Sigma\Pi\Sigma(n, k, d)$ circuit $C = T_1 + \cdots + T_k$ is said to be **minimal** if no proper subset of $\{T_i\}_{1 \leq i \leq k}$ sum to zero.

The circuit is said to be **simple** if there is no non-trivial common factor dividing all the $T_i$’s.

It was first shown by Dvir & Shpilka [19] that a minimal, simple $\Sigma\Pi\Sigma(n, k, d)$ identity has rank at most $\log^k d$. If $k$ is small then this is a much better bound than the trivial bound of $kd$. For larger $k$’s this bound is an overkill, and was improved to a more optimal-looking $k^3 \log d$ by Saxena & Seshadhri [43]. We will sketch the idea of the latter rank bound but first we discuss how a rank bound implies a black-box PIT algorithm.

### 5.1 Rank Bounds entail Black-box PIT

Karnin & Shpilka [32] showed that if we have a rank bound of $R(k, d)$ for minimal, simple $\Sigma\Pi\Sigma(n, k, d)$ identities then black-box PIT can be done in $\text{poly}(n, d^{R(k,d)})$ many field operations. Their idea was to come up with a small set of linear transformations such that: (1) for each non-zero $\Sigma\Pi\Sigma(n, k, d)$ circuit, at least one of the linear transformations keeps it non-zero, and (2) these linear transformations map the $n$ variables to $R(k, d)$ variables. It is easy to see that once we have such a linear transformation $\tau$, we can compose it with the given black-box for $C$ to get a new black-box computing $C'(x_1, \ldots, x_m) := C(\tau(x_1), \ldots, \tau(x_m))$. Since $C'$ now has “fewer” variables ($m = R(k, d)$) and is still of degree at most $d$, we could just apply a brute-force version of Schwartz-Zippel to test it for zero-ness. This entails evaluating $C'$ on $(d+1)^m$ points in $\mathbb{F}^m$ (an extension of it, if required), hence it gives an overall complexity of $\text{poly}(n, d^{R(k,d)})$.

These linear transformations $\tau$ are inspired from the *Fourier transform* matrix and they preserve the rank of arbitrary subspaces. The following
Lemma 5.2. Let $W_1, \ldots, W_s \subseteq \mathbb{F}^n$ be fixed subspaces, each of dimension at most $t$. Consider the linear transformation (for some $\alpha \in \mathbb{F}$),

$$\phi_{\alpha,t,n}(x_1, \ldots, x_n) := \left( ((\alpha^{i(j-1)}) \right)_{1 \leq i \leq t, 1 \leq j \leq n} \cdot [x_1 \cdots x_n]^T.$$

Then there are at most $s nt^2$ elements $\alpha \in \mathbb{F}$ for which the dimension of some $W_i$ drops, i.e.

$$\dim(\phi_{\alpha,t,n}(W_i)) < \dim(W_i).$$

Proof. The proof idea is to capture all the “bad” $\alpha$’s in a univariate equation, then its degree upper bounds their number.

Whenever the dimension of the image of $W_i$ under $\phi_{\alpha,t,n}$ drops, it means that the top-left $\dim(W_i) \times \dim(W_i)$ submatrix (of the matrix defining) $\phi_{\alpha,t,n}$ is singular. Thus, its determinant gives us a univariate equation in $\alpha$ of degree at most $nt^2$. Doing this for all $W_i$’s gives us the promised bound of $s nt^2$. 

Now, following [32], we give the construction of the subspaces $W_i$’s for the case of $\Sigma \Pi \Sigma(n, k, d)$ circuits assuming a rank bound of $R(k, d)$ for the minimal simple identities.

Theorem 5.3. Let $C$ be a $\Sigma \Pi \Sigma(n, k, d)$ circuit and $S \subseteq \mathbb{F}$ be of size at least $n^{2k^2} R(k, d)^2$. If $C$ is a nonzero circuit then there is an $\alpha \in S$ such that $\phi_{\alpha,n,R(k,d)}(C)$ is also nonzero.

Proof. Let $C = T_1 + \cdots + T_k$ be nonzero. We define its \textit{gcd part}, $\gcd(C) := \gcd(T_1, \ldots, T_k)$ and its \textit{simple part}, $\sim(C) := \frac{C}{\gcd(C)}$.

We now define some subspaces spanned by the linear polynomials appearing in $C$. These subspaces shall have the property that any linear transformation which preserves their dimensions, leaves $C$ nonzero. The subspaces are:

1. For every pair of linear forms $\ell, \ell'$ that appear in the circuit define $W_{\ell,\ell'} := \text{sp}(\ell, \ell')$, where $\text{sp}(\cdot)$ refers to the linear span, over $\mathbb{F}$, of the set.

2. For every nonempty subset $A \subseteq [k]$, define $C_A := \sum_{i \in A} T_i$ and let $r_A := \min\{R(k, d), \text{rank}(\sim(C_A))\}$. Let $W_A$ be the subspace spanned by $r_A$ independent linear polynomials that appear in $\sim(C_A)$.

Notice that the number of such subspaces is strictly less than $s := (k^2 d^2 + 2^k)$. The claim is that for any linear transformation $\tau = \phi_{\alpha,n,R(k,d)}$ that preserves
(the rank of) all these subspaces, also satisfies $\tau(C) \neq 0$. We will prove this by contradiction.

The first observation is that such a $\tau$ cannot map two linear functions, that appear in the circuit, to the same linear function since it preserves $W_{i,i'}$'s. Hence the simple part does not reduce further under $\tau$, i.e. we have $\text{sim}(\tau(C_A)) = \tau(\text{sim}(C_A))$. So we can assume wlog that $C$ (and hence $\tau(C)$ as well) is simple.

If $\tau(C) = 0$ then the rank bound entails that either $\tau(C)$ is not minimal or the rank($\tau(C)$) $< R(k,d)$. If the latter is the case then rank($C$) $< R(k,d)$, but then $\tau$ will preserve $W_{[k]}$, which means that the circuit $C$ is itself zero. This contradicts the hypothesis.

The other case then is: $\tau(C)$ is zero but not minimal. Let $A$ be a minimal subset such that $\tau(C_A) = 0$ with $C_A \neq 0$. Hence $\tau(\text{sim}(C_A))$ is a simple, minimal and zero circuit, therefore it is of rank less than $R(k,d)$. But then $\tau$ will preserve the rank of $W_A$, which together with $\text{sim}(C_A) \neq 0$ means that $\tau(\text{sim}(C_A)) \neq 0$. This is again a contradiction.

It is now a consequence of Lemma 5.2 that we will find such a $\tau$ if we try out $n R(k,d)^2$ many $\alpha$'s. This finishes the proof.

Algorithm & Complexity. The final black-box PIT algorithm is: given an access to a $\Sigma\Pi\Sigma(n,k,d)$ circuit $C$, try out $n 2^k d^2 R(k,d)^2$ many $\alpha$'s from the field (or its extension) and consider $C'(x_1, \ldots, x_{R(k,d)}) := \phi_{\alpha,n,R(k,d)}(C)$. Evaluate each such $C'$ on $(d + 1)^{R(k,d)}$ points on $\mathbb{F}^{R(k,d)}$, and announce $C$ to be zero iff all these evaluations are zero. It is now evident that this is a valid algorithm and it requires “only” $\text{poly}(n, 2^k, d^{R(k,d)})$ many $\mathbb{F}$ operations.

5.2 An almost Optimal Rank Bound

The best known rank bound for minimal, simple $\Sigma\Pi\Sigma(n,k,d)$ identities is $k^3 \log d$ [43]. It is also close to optimal as there are identities of rank $\Omega(k \log d)$ [31, 43]. This rank bound holds for any field. For special fields there is scope for improvement, for example Kayal & Saraf [34] showed a rank bound of $k^6$ over reals using real geometry. Note that it is independent of $d$. Both kinds of rank bounds have quite involved proofs and tend to have a strong combinatorial flavor. We will only exhibit the basic ideas of the two rank bounds by working with the “toy” example of top fanin 3. Fortunately, these basic ideas extend to the higher fanins by developing the higher-dimensional generalizations.

The $k^3 \log d$ rank bound of Saxena & Seshadhri [43] hinges on a combinatorial doubling argument. It is best visible in the top fanin 3 case and proves a sharp upper bound of $(\log d + 2)$, we prove it next.
Suppose $C = T_1 + T_2 + T_3 = 0$ is a minimal, simple $\Sigma \Pi \Sigma (n, 3, d)$ identity over some field $\mathbb{F}$. We will look at $C$ modulo various linear forms that occur in the multiplication gate $T_1$. This reflects a dependency between the forms occurring in $T_2$ and $T_3$. For instance, pick a (nonzero) linear form $q$ from $T_1$ and consider $C \pmod{q}$ which gives $T_2 + T_3 = 0 \pmod{q}$. By unique factorization of polynomials modulo $q$ this gives us a bijection $\pi$ between the forms of $T_2$ with those in $T_3$, which we call a $q$-matching between $T_2, T_3$.

**Definition 5.4. (Matchings)** Let $U, V$ be two lists of linear forms and $I$ be a form. An $I$-matching $\pi$ between $U, V$ is a bijection $\pi$ between lists $U, V$ such that: for all $\ell \in U$, $\pi(\ell) = c\ell + v$ for some $c \in \mathbb{F}^*$ and $v \in sp(I)$.

Now as we pick different $q$'s we get different matchings between $T_2, T_3$. The interesting property is that there cannot be too many such matchings if we only pick linearly independent $q$'s. This we prove in the following lemma and it immediately gives a sharp rank bound for $C$.

**Lemma 5.5.** Let $U, V$ be two lists of linear forms each of size $d > 0$ and $I_1, \ldots, I_r$ be linearly independent linear forms such that for all $i \in [r]$, there is an $I_i$-matching $\pi_i$ between $U, V$. If $r > (\log d + 2)$ then $U, V$ are similar lists (up to constant factors).

**Proof.** For contradiction assume that $r > (\log d + 2)$ but $U, V$ are not similar lists. In that case we can assume wlog that $U$ and $V$ are coprime lists, i.e. there is no linear form that occurs (up to constant factors) in both the lists.

The proof is in the form of a combinatorial process that happens on a bipartite graph. The graph $G = (U, V, E)$ has vertices labelled with the respective forms. The various $\pi_i$’s can be seen as bipartite matchings of $G$. For each $\pi_i$ and each $\ell \in U$, we add an (undirected) edge tagged with $I_i$ between the vertices $\ell$ and $\pi_i(\ell)$. There may be many tagged edges between a pair of vertices\(^1\). We call $\pi_i(\ell)$ the $I_i$-neighbor of $\ell$ (and vice versa). Abusing notation, we use vertex to refer to a form in $U \cup V$. We denote $\bigcup_{1 \leq i \leq r} I_i$ by $J_r$.

We will show that there cannot be more than $(\log d + 2)$ such perfect matchings in $G$. The proof is done by following an iterative process that has $r$ phases, one for each $I_i$. We maintain a partial basis for the forms in $U \cup V$ which will be updated iteratively. This basis is denoted by the set $B$. The goal is to completely span the forms $U \cup V$ using the forms $I_i$'s.

We start with an empty $B$ and initialize by adding some $\ell \in U$ to $B$. In the $i$th round, we will add the form $I_i$ to $B$. All forms of $U \cup V$ in $sp(\{\ell\} \cup J_i)$ are now spanned. We then proceed to the next round. To introduce some

\(^1\) It can be shown, using the independence of $I_i$’s, that an edge can have at most two distinct tags.
The forms \( \ell \) of \( \pi \) that achieve this rank bound. It was first constructed by Kayal & Saxena.

**Claim 5.6.** The forms \( \{\ell\}, I_1, \ldots, I_{w-1} \) are independent and the subspaces: \( sp\{\ell\} \cup J_0 \), \( sp(I_{w+1}), \ldots, sp(I_r) \) are independent.

**Proof of Claim 5.6.** The forms \( \{\ell\}, I_1, \ldots, I_{w-1} \) are independent by the minimality of \( i_0 \).

As \( I_1, \ldots, I_{w} \) are independent but \( \{\ell\}, I_1, \ldots, I_{w} \) are not, we deduce that \( \ell \in sp(J_0) \). Thus, the subspace \( sp\{\ell\} \cup J_0 = sp(J_0) \) is independent to the forms \( I_{w+1}, \ldots, I_r \) by the independence of \( I_1, \ldots, I_r \) \( \square \)

We shall now show that for \( i \notin \{1, i_0\} \), the number of green vertices doubles in the \( i \)-th round. Let \( \ell' \) be a green vertex, say in \( U \), at the end of the \( (i-1) \)-th round (at that point \( B = \{\ell\} \cup J_{i-1} \)). Consider the \( I_r \)-neighbor of \( \ell' \). This is in \( V \) and is equal to \( (c\ell' + v) \) where \( c \in F^* \) and \( v \) is a non-zero element in \( sp(I) \) (since \( U, V \) are coprime). If this neighbor is green, then \( v \) would be a linear combination of two green forms, implying \( v \in sp(B) \). But \( I_r \) is independent to \( B \), implying \( v \in sp(B) \cap sp(I) = \{0\} \) which is a contradiction. Therefore, the \( I_r \)-neighbor of any green vertex is not green.

On adding \( I_r \) to \( B \), the number of green vertices doubles (for at least \( (r - 2) \) rounds).

We started off with one green vertex \( \ell \), and lists \( U, V \) each of size \( d \). Thus, this doubling can happen at most \( \lg d \) times, implying that \( (r - 2) \leq \lg d \). This is a contradiction, implying that \( U, V \) are indeed similar lists. \( \square \)

The above lemma immediately implies a rank bound for our identity \( C \). As \( T_2, T_3 \) are coprime multiplication terms (by simplicity of \( C \)) the number of linearly independent forms \( q \) in \( T_1 \) can be at most \( (\lg d + 2) \). Repeating this argument wrt \( T_2 \) and \( T_3 \) proves that \( \text{rank}(C) = O(\lg d) \). Interestingly, the combinatorial procedure in the proof of Lemma 5.5 also suggests an identity that achieves this rank bound. It was first constructed by Kayal & Saxena.
\[ C(x_1, \ldots, x_r) := \prod_{b_1, \ldots, b_{r-1} \in \mathbb{F}_2, b_1 + \cdots + b_{r-1} \equiv 1} (b_1x_1 + \cdots + b_{r-1}x_{r-1}) + \prod_{b_1, \ldots, b_{r-1} \in \mathbb{F}_2, b_1 + \cdots + b_{r-1} \equiv 0} (x_r + b_1x_1 + \cdots + b_{r-1}x_{r-1}) \]

It can be seen that, over \( \mathbb{F}_2 \), \( C \) is a simple and minimal \( \Sigma\Pi\Sigma \) zero circuit of degree \( d = 2^{r-2} \) with \( k = 3 \) multiplication terms and \( \text{rank}(C) = r = \lg d + 2 \).

**In General.** The above description gives a fair snapshot of the general rank bound. For a minimal, simple \( \Sigma\Pi\Sigma(n, k, d) \) identity \( C = T_1 + \cdots + T_k \), we need to consider form-ideals \( I = (\ell_1, \ldots, \ell_{k-2}) \), where form \( \ell_i \) occurs in \( T_i \). \( C \) modulo \( I \) then gives us \( I \)-matchings between \( T_{k-1}, T_k \). If we look at such matchings modulo several linearly independent form-ideals \( I \)'s then a generalization of Lemma 5.5 says that there can be at most \( O(\lg d) \) independent form-ideals. Since each form-ideal \( I \) itself contains \( (k-2) \) independent forms, this suggests an overall rank bound of \( O(k \lg d) \). This proof idea when formalized gets into several problems, but can be salvaged to prove a rank bound of \( k^3 \lg d \). The highest rank (minimal, simple) identities known are constructed using Equation 5.1, and have rank \( \Omega(k \lg d) \). Thus, there is a slight gap in our understanding of rank.

### 5.3 A Rank Bound over Reals

The high rank identities that we saw in the last section are over fields with non-zero characteristic. When one tries to construct \( \Sigma\Pi\Sigma(n, k, d) \) identities over zero characteristic fields, say rationals, one feels that no matter how large degree \( d \) is, the rank grows only like \( k \). It was first conjectured by Dvir & Shpilka \[19\] that the rank of minimal simple \( \Sigma\Pi\Sigma(n, k, d) \) identities over zero characteristic fields should be only \( O(k) \). A weak form of this conjecture was shown true by Kayal & Saraf \[34\]. They proved a rank bound of \( k^k \) over the reals (\( \mathbb{R} \)). It is not trivial even for \( \text{fanin} = 3 \). So we give below the proof for that case and then only state its generalization.

Let \( C = T_1 + T_2 + T_3 = 0 \) be a minimal, simple \( \Sigma\Pi\Sigma(n, 3, d) \) identity over \( \mathbb{R} \). Suppose it has rank \( (r + 1) \). We identify every linear form \( \ell \) in \( C \) with the corresponding point in \( \mathbb{R}^r \). This form-to-point correspondence is just going to the projective space, roughly, a form \( (a_1x_1 + \cdots + a_{r+1}x_{r+1}) \)
is mapped to the point \( \left( \frac{a_1}{a_{r+1}}, \ldots, \frac{a_r}{a_{r+1}} \right) \) in \( \mathbb{R}^r \). This mapping gives us sets of points \( A_1, A_2, A_3 \) corresponding to the linear forms occurring in \( T_1, T_2, T_3 \) respectively. Furthermore for any forms \( \ell_1, \ell_2 \) occurring in \( T_1, T_2 \) respectively, \( C = 0 \) modulo \((\ell_1, \ell_2)\), implying that there exists a linear form \( \ell_3 \in \text{sp}(\ell_1, \ell_2) \) that occurs in \( T_3 \). This means that any line passing through a point in \( A_1 \) and a point in \( A_2 \), also passes through a point in \( A_3 \). By symmetry this means that any line passing through two of the sets \( A_1, A_2, A_3 \) also passes through the third! Such sets \( A_1, A_2, A_3 \subset \mathbb{R}^r \) are rather special and we will show below, following [20], that their existence implies \( r \leq 3 \). The proof is based on a famous theorem in incidence geometry - Sylvester-Gallai theorem.

**Theorem 5.7 (Sylvester-Gallai).** Given a finite number of non-collinear points \( S \) in the plane \( \mathbb{R}^2 \), there always exists a line which passes through exactly two points in \( S \).

**Proof.** The simple proof below is due to Kelly (see the survey by Borwein & Moser [13]).

Define a connecting line to be a line which contains at least two points from \( S \). For contradiction assume that every connecting line has a third point from \( S \). Let \((P, \ell)\) be a point and a connecting line pair that are the smallest nonzero distance apart amongst all such point-line pairs.

The line \( \ell \) goes through at least three points of \( S \). Drop a perpendicular from \( P \) to \( \ell \), there must be two points on the same side of the perpendicular (one might be exactly on the intersection of the perpendicular with \( \ell \)). Call the point closer to the perpendicular \( B \), and the farther point \( C \). Draw the line \( m \) connecting \( P \) to \( C \). Then the distance from \( B \) to \( m \) is smaller than the distance from \( P \) to \( \ell \), which is a contradiction! One way to see this is to notice that the right triangle with hypotenuse \( BC \) is similar and contained in the right triangle with hypotenuse \( PC \). This contradiction implies that there cannot be a nonzero distance between point-line pairs, thus every point must be at distance 0 from every connecting line, or in other words, every point must lie on the same line. But as \( S \) was non-collinear, we finally deduce that there exists a connecting line with exactly two points.

\[ \square \]
The Bulletin of the EATCS

Let us go back to our special sets $A_1, A_2, A_3 \subset \mathbb{R}^r$ obtained from the identity $C$ and assume (for contradiction) that $r = 4$. Pick points $p_1, p_2$ from $A_1, A_2$ respectively and consider the following pencil of planes,

$$
P := \{sp(p_1 - q, p_2 - q, q) \mid q \in A_1 \cup A_2 \cup A_3\}.
$$

Notice that $P$ consists of planes (formally 2-flats) in the space $\mathbb{R}^4$ and all of them contain the line (1-flat) joining the points $p_1, p_2$. The dual of this pencil of planes would give us a corresponding pencil of lines $L$ in $\mathbb{R}^3$. Now if we look at a section of $L$ cut by a plane in general position, we see $|L|$ non-collinear points. By Theorem 5.7 there exists a line passing through exactly two of these points, which means there exists a plane in $\mathbb{R}^3$ containing exactly two of the lines in $L$, which finally means that there exists a 3-flat in $\mathbb{R}^4$ containing exactly two of the planes in $P$. Let us denote this 3-flat by $H$ and the two planes it contains by $H_1, H_2$. Say $H_1, H_2$ are affine spans of the three points $(p_1, p_2, q_1), (p_1, p_2, q_2)$ respectively. Now if $q_1, q_2$ are not in the same $A_i$ then the line joining them (so “across” $H_1, H_2$ should pass through a point in the third set $A_j$. But that is impossible as $H$ contains only the planes $H_1, H_2$ from $P$. Thus, $q_1, q_2$ have to be in the same $A_i$, say $A_1$. But then look at the line joining $q_1, p_2$, it has to contain a point from $A_2$, say $q_3$, which will of course be in $H_1$. The line joining $q_3, q_2$ (so “across” $H_1, H_2$) should pass through a point in the third set $A_2$, which is again impossible.

This contradiction shows that our assumption $r = 4$ cannot hold, in fact, the above contradiction appears as long as $r \geq 4$. Thus, $r$ can be at most 3. This gives us a rank bound of 4 for simple $\Sigma\Pi\Sigma(n, 3, d)$ identities over reals. Interestingly, this bound is tight and there is a unique (upto transformations, see [14]) identity of rank 4 over $\mathbb{R}$:

$$
x_1 x_2 x_3 (2y + x_1 + x_2 + x_3) - (y + x_1)(y + x_2)(y + x_3)(y + x_1 + x_2 + x_3) + y(y + x_1 + x_2)(y + x_2 + x_3)(y + x_1 + x_3) = 0.
$$

In General. The above idea extends to higher fanins, but the rank bound obtained is “weaker”. Suppose $C = T_1 + \cdots + T_k = 0$ is a minimal, simple $\Sigma\Pi\Sigma(n, k, d)$ identity over $\mathbb{R}$. Instead of working in $\mathbb{R}^4$ and applying Sylvester-Gallai theorem as above, we now have to work in a much bigger space of $\mathbb{R}^m$ (roughly $m = k^4$) and use a higher-dimensional generalization of Sylvester-Gallai theorem that says:

**Theorem 5.8.** ([25], [11]) Let $S$ be a finite set of points spanning an affine space $V \subseteq \mathbb{R}^n$ such that $\dim(V) \geq 2t$. Then, there exist $(t + 1)$ points in $S$ that span a $t$-dimensional affine space $H \subset V$ such that $|H \cap S| = t + 1$. 

69
(Note that putting \( t = 1 \) above gives the statement of Sylvester-Gallai theorem.) In our case \( S \) is taken to be the set of points corresponding to all the forms that appear in \( C \). Then \( \dim(V) = \text{rank}(C) \), which we assume large enough, say \( k^k \), to derive a contradiction. Kayal & Saraf [34] now use Theorem 5.8 to identify a subspace of \( V \) and its decomposition (analogous to \( H \) and its “decomposition” \( H_1, H_2 \) above), and from that deduce the existence of a linear form \( \ell \) in \( C \) such that \( C(\text{mod } \ell) \) has a minimal, simple \( \text{sub-identity} \) of rank at least \( (k - 1)k^{k-1} \). This gives the promised contradiction as \( C(\text{mod } \ell) \), and hence the sub-identity, has a smaller fanin.

**Open.** It would be interesting to improve the above rank bound: (1) to other zero characteristic fields, (2) to a more optimal-looking bound \( O(k) \).

It is known that Sylvester-Gallai theorem 5.7 is false in \( \mathbb{C}^2 \) (cubic curves give counter examples [18]). Nevertheless, it is known to hold in the following sense [28, 21]: Given a finite number of non-coplanar points \( S \) in \( \mathbb{C}^3 \), there always exists a line which passes through exactly two points in \( S \). This immediately gives us a rank bound of 5 (unlike 4 before) for simple \( \Sigma \Pi \Sigma(n, 3, d) \) identities over \( \mathbb{C} \). Unfortunately, a higher-dimensional generalization of this version is not known. A natural conjecture for it would be:

**Conjecture 5.9.** Let \( S \) be a finite set of points spanning an affine space \( V \subseteq \mathbb{C}^n \) such that \( \dim(V) \geq 3t \). Then there exist \( (t + 1) \) points in \( S \) that span a \( t \) dimensional affine space \( H \subset V \) such that \( |H \cap S| = t + 1 \).

(Currently, a proof exists only for \( t = 1 \).)

### 6 Depth-4 PIT

We know that depth-4 case of PIT has direct relations to more general cases of PIT. But currently we have little understanding of depth-4 circuits. For example, we do not even know how to test \( f_1 \cdots f_m = g_1 \cdots g_m \) where \( f_i \)'s and \( g_i \)'s are multivariate polynomials given in **fully expanded form** (also called the sparse representation). Here we will discuss two simple ideas that solve PIT for certain restricted forms of depth-4 circuits.

**Noncommuting Idea.** The first idea is easiest to see on depth-4 circuits \( C \) whose multiplication gates have **unmixed** variables, i.e. \( C(x_1, \ldots, x_n) = M_1 + \cdots + M_k \), where for all \( i \in [k] \), \( M_i = f_{i,1}(x_1) \cdots f_{i,n}(x_n) \), where each \( f_{i,j} \) is a univariate polynomial given in the sparse representation and is of degree at most \( d \). Note that “expanding out” \( M_i \) in a brute force way could potentially produce \( d^n \) monomials and hence is not recommended!

Interestingly, Raz & Shpilkam formulated a “controlled” way of doing this expansion using basic linear algebra. Their idea was to compute
$f_{i,1}(x_1)f_{i,2}(x_2)$, for all $i \in [k]$. View each of these polynomials as vectors in a natural way, i.e. each coefficient is a coordinate of the vector. Thus we have $k$ vectors $V := \{v_1, \ldots, v_k\}$ in a space of dimension at most $d^2$. Pick some maximal subset of $V$ that has linearly independent vectors, say they are $v_1, \ldots, v_{\ell}$ ($1 \leq \ell \leq k$). Now we form the circuit $C_1$ from $C$ by replacing $f_{i,1}(x_1)f_{i,2}(x_2)$, for all $i \in [\ell]$, by a fresh variable $z_{1,i}$. For the other $i$’s, replace $f_{i,1}(x_1)f_{i,2}(x_2)$ by the same linear combination of $\{z_{1,1}, \ldots, z_{1,\ell}\}$ as the one that expresses $v_i$ in terms of $\{v_1, \ldots, v_{\ell}\}$. It is easy to verify that $C(x_1, \ldots, x_n) = 0$ iff $C_1(z_{1,1}, \ldots, z_{1,\ell}, x_3, \ldots, x_n) = 0$. Thus this one round reduced the number of factors in each $M_i$ by one at the cost of increasing the number of variables by $O(k)$. If we repeat this round $(n-2)$ more times then instead of $M_i$’s we would have just linear forms and the number of variables would be $O(nk)$. As the linear algebra in each round requires just \(poly(ndk)\) operations, and testing the zero-ness of the circuit after the last round is trivial, we get an overall complexity of \(poly(ndk)\) field operations.

It can be easily verified that the technique above is applicable in several other cases, in particular: (1) when $C$ is a \textit{set-multilinear} formula, i.e. each multiplication gate $M_i$ has inputs that are polynomials in disjoint variables $S_1, \ldots, S_\ell$ fixed such that $S_1 \cup \cdots \cup S_\ell = \{x_1, \ldots, x_n\}$. (2) more generally, when $C$ is a \textit{noncommutative} formula, i.e. variables $x_1, \ldots, x_n$ do not commute w.r.t. multiplication.

\textbf{Powering Idea.} The second idea is to consider depth-4 circuits $C$ whose multiplication gates just do powering, i.e. $C(x_1, \ldots, x_n) = M_1 + \cdots + M_k$, where for all $i \in [k]$, $M_i = \alpha_i \cdot (f_{i,1}(x_1) + \cdots + f_{i,n}(x_n))^\epsilon_i$, where each $f_{i,j}$ is a univariate polynomial given in the sparse representation and is of degree at most $d$, $\alpha_i \in \mathbb{F}$ and $\epsilon_i \in \mathbb{N}$. Again note that “expanding out” $M_i$ in a brute force way could potentially produce more than $\binom{n + \epsilon_i}{\epsilon_i}$ monomials. Interestingly, although this case looks to be on the other extreme of the “unmixed variable” case discussed above, a reduction of the former to the latter was given by Saxena [38]. The following lemma gives the main transformation:

\textbf{Lemma 6.1.} Let $g_1(x_1), \ldots, g_n(x_n)$ be univariate polynomials of degree at most $d$, over a field $\mathbb{F}$ of zero characteristic. Then we can compute univariate polynomials $h_{i,j}$’s in \(poly(n d)\) field operations such that for $t = (n + 1)$:

$$(g_1(x_1) + \cdots + g_n(x_n))^a = \sum_{i=1}^{t} h_{i,1}(x_1) \cdots h_{i,n}(x_n)$$

\textbf{Proof.} We will prove this using the formal power series: $exp(x) = 1 + x + \frac{x^2}{2!} + \cdots$, where $exp(x) = e^x$ and $e$ is the base of natural logarithm. Define the degree $a$ truncation of the series to be $E_a(x) = 1 + x + \cdots + \frac{x^a}{a!}$. We
will use the operator \([z^a]\) to extract the coefficient of \(z^a\) from a polynomial. Observe that:

\[
(a!)^{-1} \cdot (g_1(x_1) + \cdots + g_n(x_n))^a = [z^a] \exp((g_1(x_1) + \cdots + g_n(x_n)) \cdot z)
\]

\[
= [z^a] \exp(g_1(x_1)z) \cdots \exp(g_n(x_n)z)
\]

\[
= [z^a] E_a(g_1(x_1)z) \cdots E_a(g_n(x_n)z)
\]

The product \(E_a(g_1(x_1)z) \cdots E_a(g_n(x_n)z)\) can be viewed as a univariate polynomial in \(z\) of degree \(na\). Hence, its coefficient of \(z^a\) can be computed by evaluating the polynomial at \(t\) distinct points \(\alpha_1, \ldots, \alpha_t \in \mathbb{F}\) (remember \(\mathbb{F}\) is large enough) and by interpolation we can compute \(\beta_1, \ldots, \beta_t \in \mathbb{F}\) such that:

\[
[z^a] E_a(g_1(x_1)z) \cdots E_a(g_n(x_n)z) = \sum_{i=1}^{t} \beta_i \cdot E_a(\alpha_i g_1(x_1)) \cdots E_a(\alpha_i g_n(x_n))
\]

This can be seen as the dual form of the multiplication gate \((g_1(x_1) + \cdots + g_n(x_n))^a\). It is routine to verify that all the univariate polynomials \(E_a(\cdot)\) in the above sum can be computed in \(\text{poly}(nda)\) field operations.

Applying this lemma to all the gates \(M_i\)'s of \(C(x_1, \ldots, x_n) = M_1 + \cdots + M_k\), we convert our given circuit \(C\) to another circuit \(C'(x_1, \ldots, x_n)\) which is a sum-of-product of univariates. Such a \(C'\) can now be tested for zeroness using the noncommuting idea seen above.

The technique of Lemma 6.1 also applies to a slightly general case of \((s\) is any constant): \(C(x_1, \ldots, x_n) = \sum_{i=1}^{k} L_{i,1}^a \cdots L_{i,s}^a\) where the \(L_{i,j}\)'s are sums of univariate polynomials, i.e. for all \(i \in [k], j \in [s] \):

\(L_{i,j}(x_1, \ldots, x_n) = f_{i,j,1}(x_1) + \cdots + f_{i,j,n}(x_n)\) where \(f_{i,j,p} \in \mathbb{F}[x_p]\).

Open. The PIT algorithms in the above two restricted cases of depth-4 circuits are inherently non black-box. Are there black-box PIT algorithms for these family of depth-4 circuits? Currently, there are no black-box PIT algorithms known for any nontrivial family of depth-4 circuits.

7 General PIT and Lower Bounds

We saw in the above sections that PIT algorithms, at least the ones currently known, are quite involved and require ideas from algebra, geometry and
The Bulletin of the EATCS

combinatorics. This proof complexity is partially explained by the connection PIT has to certain circuit lower bounds (that are historically considered difficult to prove!). We will now discuss how a theorem like PIT $\in P$ would imply lower bounds [29], and that a black-box PIT algorithm would imply even stronger lower bounds [3].

**Implications of PIT in P.** Kabanets & Impagliazzo [29] showed that if PIT has a deterministic polynomial time algorithm then *either* NEXP (i.e. nondeterministic exponential time class) does not have polynomial sized boolean circuits (i.e. $P/poly$ class) *or* the perm function (i.e. permanent of a square matrix of rationals) does not have polynomial sized arithmetic circuits (i.e. $AlgP/poly$ class). Note that both the claims $NEXP \not\subseteq P/poly$ and $perm \not\in AlgP/poly$ are conjectured to be true by “most” people. So the main attraction of the following theorem is that it connects an algorithmic conjecture like PIT $\in P$ with these lower bound conjectures.

**Theorem 7.1.** ([29]) *If low-degree-PIT $\in P$ then:* $NEXP \not\subseteq P/poly$ *or* $perm \not\in AlgP/poly$.

**Proof Sketch:** The proof is by contradiction, so we assume:

1. low-degree-PIT $\in P$
2. $perm \in AlgP/poly$
3. $NEXP \subseteq P/poly$

Assumptions (1) and (2) imply that, one can guess an arithmetic circuit $C$ and then actually check whether $C = perm$ using PIT. As permanent of an $m \times m$ matrix has degree $m$, one only needs to guess $C$ of depth $O(\log m)$ and multiplication fanin 2 [6], then do low degree PIT. The part where we check $C = perm$ using PIT, actually makes use of the downward self-reducibility of the permanent function, i.e. $perm$ of an $m \times m$ matrix can be expressed as the sum of $m$ permanents each of $(m-1) \times (m-1)$ submatrices. Finally,

4. $P^{perm} \subseteq NP$.

Now it is known that assumption (3) implies $NEXP \subseteq P^{perm}$ [26]. This together with deduction (4) implies that $NEXP \subseteq NP$, which is a contradiction as there are classical *diagonalization* methods proving NEXP different from NP [40].

One might wonder whether this theorem has a converse. As a step in that direction, it was shown in [29]: if permanent has superpolynomial arithmetic circuit complexity then PIT has a subexponential time algorithm. The idea is
to apply a hard function (here permanent) on the designs defined by Nisan & Wigderson [36], and evaluate the given circuit \( C \) at the resulting point. The claim is that this evaluation is zero iff \( C \) is a zero circuit. Thus, hard algebraic functions give black-box PIT algorithms!

**Implications of black-box PIT.** As we have mentioned before, black-box PIT algorithms seem to require a very good understanding of the circuit family and hence should, intuitively, also imply what that circuit family cannot compute! It is interesting, this intuition can also be proven formally, as we will now show following Agrawal [3].

A black-box PIT algorithm is only allowed to evaluate a given circuit \( C(x_1, \ldots, x_n) \) at points in the extensions of the given field \( F \). Thus, it seems reasonable to assume that such an algorithm just “feeds in” \( x_i = f_i(y) \) (mod \( g(y) \)) for all \( i \in [n] \), where \( f_i \)'s and \( g \) are univariate polynomials of a “small” degree \( 2^{\ell(n)} \). Note that the algorithm feeds these polynomials to every input circuit \( C(x_1, \ldots, x_n) \), only assuming that \( C \) has a size bound of \((\text{wlog}) \ n \). Clearly, the time complexity of such a black-box PIT algorithm is dominated by \( 2^{\ell(n)} \), and the time taken to actually construct \( f_i \)'s and \( g \). This motivates the definition of a *pseudo-random generator (prg)* for arithmetic circuits.

**Definition 7.2.** Fix a field \( F \). A function \( f : \mathbb{N} \rightarrow (F[y])^n \) is called an efficient \((\ell(n), n)\)-prg if,

- \( f(n) \in (F[y])^{n+1} \) for all \( n > 0 \).
- \( f(n) = (f_1(y), \ldots, f_n(y), g(y)) \) where the polynomials \( f_i \)'s and \( g \) are of degree at most \( 2^{\ell(n)} \), and are also constructible in \( \text{poly}(2^{\ell(n)}) \) time.
- For any circuit \( C(x_1, \ldots, x_n) \) of size at most \( n \), \( C(x_1, \ldots, x_n) = 0 \) iff \( C(f_1(y), \ldots, f_n(y)) = 0 \) (mod \( g(y) \)).

If we drop the requirement of efficient constructibility then such functions \( f \), for any \( \ell(n) = \Omega(\lg n) \), can be easily shown to exist using the Schwartz-Zippel lemma. On the other hand it can be seen, by the methods of Section 2, that efficient ones for \( \ell(n) = O(n^2) \) exist. The really interesting cases for us are in between, and so we will always assume \( \ell(n) = \Omega(\lg n) \) and \( \ell(n) = o(n) \). The existence of an efficient \((\ell(n), n)\)-prg immediately gives a black-box PIT algorithm with time complexity \( \text{poly}(2^{\ell(n)}) \). Thus, to completely solve PIT we “just” need an efficient \((O(\lg n), n)\)-prg. We now show that such a prg implies arithmetic circuit lower bounds (that are beyond the scope of current methods).
The Bulletin of the EATCS

**Theorem 7.3.** ([3]) If there is an efficient \((\ell(n), n)\)-prg. Then there is a multilinear polynomial that is \(\text{poly}(2^{\ell(n)})\) time computable but has no circuits of size \(n\).

**Proof Sketch:** Let \(f(n) = (f_1, \ldots, f_n(y), g(y))\) be an efficient \((\ell(n), n)\)-prg. Let \(m := \ell(n)\). In the interesting case of \(\ell(n) = o(n)\), we have \(n > 2m\). We define a polynomial \(q_f(x_1, \ldots, x_{2m})\) as:

\[ q_f(x_1, \ldots, x_{2m}) := \sum_{S \subseteq [1, 2m]} c_S \cdot \prod_{i \in S} x_i. \]

Where the coefficient \(c\)'s are picked such that they satisfy:

\[ q_f(f_1(y), \ldots, f_{2m}(y)) = \sum_{S \subseteq [1, 2m]} c_S \cdot \prod_{i \in S} f_i(y) = 0. \]

The existence of such coefficient \(c\)'s can be seen by comparing the degree of the above equation in \(y\) and the number of the unknowns. Furthermore, the polynomial \(q_f\) can be computed by solving a system of \(\text{poly}(2^m)\) linear equations in \(\text{poly}(2^m)\) variables over the field \(\mathbb{F}\). Each of these equations can be computed in time \(\text{poly}(2^m)\) using the computability of \(f\). Therefore, \(q_f\) can be computed in time \(\text{poly}(2^m)\).

Now suppose \(q_f\) can be computed by a circuit \(C\) of size \(n\). By the definition of polynomial \(q_f\), it follows that \(C(f_1(y), f_2(y), \ldots, f_{2m}(y)) = 0\). On the other hand, the size of the circuit \(C\) is \(n\) and it computes a nonzero polynomial. This contradicts to \(f\) being a prg. Hence \(q_f(x_1, \ldots, x_{2\ell(n)})\) is a multilinear polynomial computable in \(\text{poly}(2^{\ell(n)})\) time but not by \(n\)-sized circuits.

The way \(q_f\) is defined above has the nice property that it can be expressed as the permanent of a “small” matrix [4]. Thus the existence of an efficient prg \(f\) would not only give a “hard” polynomial \(q_f\) but indeed prove the hardness of permanent function!

**References**


In light of its flexibility and its intuitive appeal, Structural Operational Semantics is by now one of the most popular approaches to giving the semantics of programming and specification languages. Over the last twenty five years, researchers in concurrency theory have developed a number of general results guaranteeing semantic properties for whole classes of languages, provided their operational semantics is given by rules that meet certain syntactic constraints. Some of these results link operational semantics with algebraic semantics in that they aim either at guaranteeing the validity of general algebraic properties of language constructs "by design" or at generating valid algebraic equations from the operational specification of language constructs. This contribution to the Concurrency Column offers a survey of some of the methods that have been developed in the literature on this line of research. It provides a glimpse of an ongoing research programme, which I hope will be of interest to the readers of this column.

On a different note, I am happy to report that CONCUR 2009 in Bologna was a resounding success. The scientific programme for the main conference and its affiliated events was of very high quality and the organization was outstanding. Many thanks to Mario Bravetti, Gianluigi Zavattaro and their team for organizing a splendid twentieth edition of CONCUR. I encourage the readers of the Concurrency Column to read the proceedings of CONCUR 2009. There is plenty of food for thought on those pages and this bodes well for the future of concurrency theory and of the CONCUR conference series.
Algebraic Properties for Free!*  
Luca Aceto, Anna Ingolfsdottir  
School of Computer Science, Reykjavik University,  
Kringlan 1, IS-103, Reykjavik, Iceland  
MohammadReza Mousavi, Michel A. Reniers  
Department of Computer Science,  
Eindhoven University of Technology,  
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands  

Abstract  
Algebraic properties specify some natural properties of programming and specification constructs. This paper provides an overview of techniques to guarantee or generate algebraic properties of language constructs by investigating the syntactic shape of the deduction rules defining their operational semantics.

1 Introduction  
Programming and specification languages are defined in terms of a syntax and a semantics. The syntax of a language specifies the grammatical structure of well-formed programs, while its semantics defines the intended meaning of syntactically valid programs. The definition of the syntax of a language is usually given in terms of a grammar, most often in the Backus-Naur Form or one of its variants. The syntax of a language can also be seen as an algebraic structure with programming or specification constructs as operators, or function symbols, that allow one to construct composite program fragments from their components. This paves the way to the use of algebraic methods for reasoning about programs. In fact, when designing a language, a designer usually has certain algebraic properties of operators in mind, e.g., $x_0; \text{skip} = x_0$. Such properties can either be validated

---

*The work of Aceto and Ingolfsdottir has been partially supported by the projects “The Equational Logic of Parallel Processes” (nr. 060013021) and “New Developments in Operational Semantics” (nr. 080039021) of the Icelandic Research Fund.
Algebraic properties specify some natural properties of function symbols from a given signature. Examples of such properties include commutativity, associativity and idempotence of binary operators, which are concisely specified, respectively, by the following equations:

\[
\begin{align*}
    f(x_0, x_1) &= f(x_1, x_0) \\
    f(f(x_0, x_1), x_2) &= f(x_0, f(x_1, x_2)) \\
    f(x_0, x_0) &= x_0.
\end{align*}
\]

In the context of programming and specification languages, algebraic properties prescribe the equivalences that must be respected by all the semantic models of a language. In other words, they ensure that the semantics of certain composition schemes must preserve the “intended behaviour” of programs. The most popular way to define the operational semantics (hence the “behaviour”) of computer languages is Structural Operational Semantics (SOS) [37, 36, 24]. Hence, it makes sense to establish generic methods that can guarantee and/or derive algebraic properties based on the SOS specifications of language constructs. This paper provides an overview of some of the available techniques to this end. In keeping with the expository nature of this article, we do not present the results in full generality since this would obscure the main message. However, we provide references to the literature for the readers interested in the technical details and generalizations of the results we cover in this survey.

The rest of this paper is organized as follows. In Section 2, we present some standard notions from the meta-theory of SOS. In Sections 3, 4 and 5, respectively, we report on work that aims at isolating sufficient syntactic conditions over SOS specifications that guarantee the validity of commutativity, associativity and idempotence axioms from SOS specifications. In Section 6, we present a survey of existing meta-theorems to generate sound and ground-complete axiom systems from SOS specifications. Since these axiom systems are ground-complete, all ground instances of abstract algebraic properties such as commutativity, associativity and idempotence can be derived from them. However, the generic meta-theorems presented in Section 6 do not derive such axioms explicitly. Hence, the techniques presented in Sections 3-5 can be complementary to those presented in Section 6. We get back to this issue in our concluding remarks and open problems, which are presented in Section 7.

2 Preliminaries

In this section we review, for the sake of completeness, some standard definitions from the meta-theory of SOS that will be used in the remainder of the paper. We refer the interested reader to [7, 32] for further details.
Definition 1 (Signature and terms). We let $V$ represent an infinite set of variables with typical members $x, x', x_1, y, y', y_1, \ldots$. A signature $\Sigma$ is a set of function symbols, each with a fixed arity. We call these symbols operators and usually represent them by $f, g, \ldots$. An operator with arity zero is called a constant. We define the set $T(\Sigma)$ of terms over $\Sigma$ as the smallest set satisfying the following constraints.

- A variable $x \in V$ is a term.
- If $f \in \Sigma$ has arity $n$ and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

We use $t, t', t_1, \ldots$ to range over terms. We write $t_1 \equiv t_2$ if $t_1$ and $t_2$ are syntactically equal. The function $\text{vars} : T(\Sigma) \to 2^V$ gives the set of variables appearing in a term. The set $\mathbb{C}(\Sigma) \subseteq T(\Sigma)$ is the set of closed terms, i.e., terms that contain no variables. We use $p, p', p_1, \ldots$ to range over closed terms. A substitution $\sigma$ is a function of type $V \to T(\Sigma)$. We extend the domain of substitutions to terms homomorphically. If the range of a substitution lies in $\mathbb{C}(\Sigma)$, we say that it is a closed substitution.

Definition 2 (Transition System Specifications (TSS), formulae and transition systems). A transition system specification is a triplet $(\Sigma, L, D)$ where

- $\Sigma$ is a signature.
- $L$ is a set of labels. If $l \in L$, and $t, t' \in T(\Sigma)$ we say that $t \xrightarrow{l} t'$ is a (positive) formula. A formula is typically denoted by $\phi, \psi, \phi', \phi_1, \ldots$.
- $D$ is a set of deduction rules, i.e., pairs of the form $(\Phi, \phi)$ where $\Phi$ is a set of formulae and $\phi$ is a formula. We call the formulae contained in $\Phi$ the premises of the rule and $\phi$ the conclusion.

We write $\text{vars}(r)$ to denote the set of variables appearing in a deduction rule $r$. We say that a formula is closed if all of its terms are closed. Substitutions are also extended to formulae and sets of formulae in the natural way. A set of positive closed formulae is called a transition system. The transition system induced by a TSS is the set of provable positive formulae, using its deduction rules. We write $\mathcal{T} \vdash p \xrightarrow{l} p'$ if the transition $p \xrightarrow{l} p'$ is provable from the deduction rules in the TSS $\mathcal{T}$.

We often refer to a formula $t \xrightarrow{l} t'$ as a transition with $t$ being its source, $l$ its label, and $t'$ its target. Predicates over terms can be seen as transitions with a dummy target [41]. A deduction rule $(\Phi, \phi)$ is typically written as $\frac{\Phi}{\phi}$. We call a deduction rule $f$-defining when the outermost function symbol appearing in the source of its conclusion is $f$. For simplicity, we assume in the remainder of this
paper that all deduction rules are $f$-defining for some function symbol $f$, i.e., there are no deduction rules with a variable as the source of its conclusion. Note that we have confined ourselves to positive TSSs, i.e., those that only use formulae of the form $t \rightarrow t'$; most of the results presented in what follows have already been extended to the setting with negative formulae, i.e., formulae of the form $t \rightarrow$, as premises. We refer to the corresponding papers for further details.

To establish a link between the operational model and the algebraic properties, a notion of behavioural equivalence should be fixed. A very common notion of behavioural equivalence that is mentioned frequently in this paper is the following notion of strong bisimilarity [34, 28].

**Definition 3 (Strong Bisimilarity).** Let $T$ be a TSS with signature $\Sigma$. A relation $R \subseteq C(\Sigma) \times C(\Sigma)$ is a strong bisimulation relation if and only if $R$ is symmetric and for all $p_0, p_1, p'_0 \in C(\Sigma)$ and $l \in L$

$$(p_0 \; R \; p_1 \land T \vdash \sigma(t_0) \rightarrow l) \Rightarrow \exists p'_1 \in C(\Sigma) (T \vdash \sigma(t_1) \rightarrow l' \land p'_0 \; R \; p'_1).$$

Two terms $p_0, p_1 \in C(\Sigma)$ are called strongly bisimilar, denoted by $T \vdash p_0 \leftrightarrow p_1$, when there exists a strong bisimulation relation $R$ such that $p_0 \; R \; p_1$.

Any equivalence relation $\sim$ over closed terms in a TSS $T$ is extended to open terms in the standard fashion, i.e., for all $t_0, t_1 \in T(\Sigma)$, the equation $t_0 \sim t_1$ holds over $T$ modulo $\sim$ if, and only if, $T \vdash \sigma(t_0) \sim \sigma(t_1)$ for each closed substitution $\sigma$.

Ideally, the notion of behavioural equivalence should coincide with the equational theory generated by a set of axioms describing the desired algebraic properties of the operators in a language. One side of this coincidence is captured by the soundness theorem, which states that all the (closed) equalities that are derivable from the axiom system using the rules of equational logic are indeed valid with respect to the particular notion of behavioural equivalence. The other side of the coincidence, called ground-completeness, states that all the valid behavioural equivalences (on closed terms) are derivable from the axiom system, as well. These concepts are formalized in what follows.

**Definition 4 (Axiom System).** An axiom system $E$ over a signature $\Sigma$ is a set of equalities of the form $t = t'$, where $t, t' \in T(\Sigma)$. A closed equality $p = p'$, for some $p, p' \in C(\Sigma)$, is derivable from $E$, denoted by $E \vdash p = p'$, if and only if it is in the smallest congruence relation on closed terms induced by the equalities in $E$.

In the context of a fixed TSS $T$, an axiom system $E$ is sound with respect to a notion of behavioural equivalence $\sim$ if and only if for all $p, p' \in C(\Sigma)$, if $E \vdash p = p'$, then it holds that $T \vdash p \sim p'$. It is ground-complete if the implication holds in the other direction.
3 Commutativity Format

Commutativity is an essential property of binary operators specifying that the order of arguments is immaterial. For process algebras, commutativity is defined with respect to a notion of behavioural equivalence. The commutativity format proposed in [31], and given below, guarantees commutativity with respect to any notion of behavioural equivalence that is specified in terms of transitions, e.g., all notions in van Glabbeek’s spectrum [21, 20] that include strong bisimilarity.

Definition 5 (Commutativity). Given a TSS and a binary operator \( f \) in its signature, \( f \) is called commutative w.r.t. \( \sim \), if the following equation is sound w.r.t. \( \sim \):

\[
f(x_0, x_1) = f(x_1, x_0).
\]

Next, we present a syntactic restriction of the tyft format of Groote and Vaandrager [23] that guarantees commutativity w.r.t. any notion of behavioural equivalence that includes strong bisimilarity.

Definition 6 (Comm-form). A transition system specification over signature \( \Sigma \) is in comm-form format with respect to a set of binary function symbols \( \text{COMM} \subseteq \Sigma \) if all its \( f \)-defining deduction rules with \( f \in \text{COMM} \) have the following form

\[
\begin{align*}
\frac{\{ x_j \xrightarrow{\text{ij}} y_{ij} \mid i \in I \}}{f(x_0, x_1) \xrightarrow{t} t}
\end{align*}
\]

where \( j \in \{0, 1\} \), \( I \) is an arbitrary index set, and variables appearing in the source of the conclusion and target of the premises are all pairwise distinct. We denote the set of premises of \((d)\) by \( H \) and the conclusion by \( c \). Moreover, for each such rule, there exist a deduction rule \((d')\) of the following form in the transition system specification

\[
\begin{align*}
\frac{H'}{f(x'_0, x'_1) \xrightarrow{t'} t'}
\end{align*}
\]

and a bijective mapping (substitution) \( h \) on variables such that

- \( h(x'_0) = x_1 \) and \( h(x'_1) = x_0 \),
- \( h(t') \sim_{cc} t \) and
- \( h(h') \in H \cup \{ c \} \), for each \( h' \in H' \),

where \( \sim_{cc} \) means equality up to swapping of arguments of operators in \( \text{COMM} \) in any context. Deduction rule \((d')\) is called the commutative mirror of \((d)\).
To put it informally, the role of substitution $\mathcal{h}$ in this definition is to account for the swapping of variables in the source of the conclusion and a possible isomorphic renaming of variables. Thus, the above format requires that when $f \in \text{COMM}$, for each $f$-defining rule, there exists a commutative mirror that en- 
ables the “same transitions” when the two arguments of $f$ are swapped. (The format presented in [31] is more general and allows for much more general types of rules and an arbitrary swapping of arguments of $n$-ary commutative function symbols. We simplified the format to facilitate the presentation here.)

**Theorem 7.** If a transition system specification is in **comm-form** format with respect to a set of binary operators COMM, then all operators in COMM are commutative with respect to any notion of behavioural equivalence that includes strong bisimilarity.

**Example 8** (Parallel Composition). A frequently occurring commutative operator is parallel composition. It appears in, amongst others, ACP [14], CCS [27], and CSP [26, 38]. Here we discuss parallel composition with communication in the style of ACP [14], of which the others are special cases.

\[
\begin{align*}
\text{(p0)} & \quad x_0 \xrightarrow{l} y_0 \\
& \quad x_0 \| x_1 \xrightarrow{l} y_0 \| x_1 \\
\text{(p1)} & \quad x_1 \xrightarrow{l} y_1 \\
& \quad x_0 \| x_1 \xrightarrow{l} x_0 \| y_1 \\
\text{(p2)} & \quad x_0 \xrightarrow{l_0} y_0 \xrightarrow{l_1} x_1 \xrightarrow{l_1} y_1 \\
& \quad x_0 \| x_1 \xrightarrow{l} y_0 \| y_1 \\
\end{align*}
\]

If the partial synchronization function comm is commutative, then the above TSS is in **comm-form** format w.r.t. the singleton set COMM = {||} and hence it follows from Theorem 7 that || is commutative.

**Example 9** (Nondeterministic Choice). Most process languages for the description of sequential and parallel systems contain a form of nondeterministic choice operator (also called alternative composition). Here we introduce nondeterministic alternative composition as present in CCS [27] and ACP [14].

\[
\begin{align*}
\text{(c0)} & \quad x_0 \xrightarrow{l} y_0 \\
& \quad x_0 + x_1 \xrightarrow{l} y_0 \\
\text{(c1)} & \quad x_1 \xrightarrow{l} y_1 \\
& \quad x_0 + x_1 \xrightarrow{l} y_1 \\
\end{align*}
\]

Using Theorem 7, we can derive that nondeterministic choice is a commutative operator w.r.t. strong bisimilarity or coarser behavioural equivalences. Both of (c0) and (c1) are in **tyft** format and, taking COMM = {+}, each is the commutative mirror of the other under the mapping $\mathcal{h}(x_0) = x_1$, $\mathcal{h}(x_1) = x_0$, $\mathcal{h}(y_1) = y_0$ and $\mathcal{h}(y_0) = y_1$. 

87
4 Associativity Format

Associativity (with respect to a given notion of equivalence), defined below, is an interesting property of binary operators, which does not lend itself easily to syntactic checks like those described in the previous section.

Definition 10 (Associativity). A binary operator \( f \in \Sigma \) is associative w.r.t. an equivalence \( \sim \) if and only if the following equation is sound w.r.t. \( \sim \):

\[
f(x_0, f(x_1, x_2)) = f(f(x_0, x_1), x_2).
\]

Because of the nesting of function symbols in associativity axioms, proofs of associativity are usually much more laborious than proofs of both congruence properties and commutativity. For example, proofs of congruence, by and large, make use of induction on the proof structure for transitions and are confined to look at the proof tree up to depth one. Thus, they can be performed by looking at deduction rules individually. Proofs of associativity, on the other hand, are usually concerned with proof trees of depth two and hence, if there are \( n \) deduction rules for a certain binary operator each with \( m \) premises, the number of case distinctions in its associativity proof are \( n^m \) in the worst case (for each deduction rule and each premise, there are \( n \) possible deduction rules responsible for the transition mentioned in the premise). This is why in [31, Section 5], two of the authors report that their initial attempt to devise a syntactic constraint for associativity did not lead to a concrete rule format. In [18], the same authors (and a co-author) developed an associativity rule format, which is given below.

This format, called ASSOC-de Simone, is a syntactic rule format that guarantees associativity of a binary operator with respect to any notion of behavioural equivalence that is specified in terms of transitions. We take the de Simone format [19] as our starting point and add a number of other ingredients, such as predicates [11], to it. Our choice of the de Simone format is motivated by the inherent complexity of associativity proofs and aims to reduce the size and the number of the proof trees as much as possible. The extensions are motivated by practical examples as illustrated in [18]. Despite the simple setting of our format, the ASSOC-de Simone format is widely applicable to most practical examples we encountered so far. Moreover, we show in [18] that dropping any of the restrictions of the format jeopardizes the meta-result, even for associativity with respect to notions of behavioural equivalence that are coarser than bisimilarity, such as trace equivalence.

Definition 11 (The ASSOC-de Simone Rule Format). Let \( \gamma : L \times L \rightarrow L \) be an associative partial function. Consider the following types of rules which are all in the de Simone format.
The Bulletin of the EATCS

1. Left-conforming rules
   \[ x_0 \xrightarrow{f} y_0 \]
   \[ f(x_0, x_1) \xrightarrow{f} f(y_0, x_1) \]

2. Right-conforming rules
   \[ x_1 \xrightarrow{f} y_1 \]
   \[ f(x_0, x_1) \xrightarrow{f} f(x_0, y_1) \]

3. Left-choice rules
   \[ x_0 \xrightarrow{f} y_0 \]
   \[ f(x_0, x_1) \xrightarrow{f} y_0 \]

4. Right-choice rules
   \[ x_1 \xrightarrow{f} y_1 \]
   \[ f(x_0, x_1) \xrightarrow{f} y_1 \]

5. Left-choice axioms
   \[ f(x_0, x_1) \xrightarrow{f} x_0 \]
   \[ f(x_0, x_1) \xrightarrow{f} y_0 \]

6. Right-choice axioms
   \[ f(x_0, x_1) \xrightarrow{f} x_1 \]

7. Communicating rules
   \[ l_0 \xrightarrow{\gamma} l_1 \]
   \[ f(x_0, x_1) \xrightarrow{\gamma} f(y_0, y_1) \]

A TSS is in the ASSOC-de Simone format with respect to \( f \in \Sigma \) when for each \( l \in L \), each \( f \)-defining rule is of a type given above and the set of all \( f \)-defining rules satisfies the following constraints. (Each proposition \( P \) in the following constraints should be read as “there exists a deduction rule of type \( P \) in the set of \( f \)-defining rules”. To avoid repeated uses of parentheses, we assume that \( \lor \) and \( \land \) take precedence over \( \Rightarrow \) and \( \Leftrightarrow \).)

1. \( 5_l \Rightarrow 2_l \land 3_l \)
2. \( 6_l \Rightarrow 1_l \land 4_l \)
3. \( 7_{(l,l')} \Rightarrow (1_l \Leftrightarrow 2_{l'}) \land (3_l \Leftrightarrow 4_{l'}) \land (2_l \Leftrightarrow 2_{l(l,l')}) \land (4_l \Leftrightarrow 4_{l(l,l')}) \land (1_{l'} \Leftrightarrow 1_{l(l,l')}) \land (3_{l'} \Leftrightarrow 3_{l(l,l')}) \)
4. \( 1_l \land 3_l \Rightarrow \exists_{l'} \gamma(l, l') = l \land 7_{(l,l')} \land 5_{l'} \land 6_{l'} \)
5. \( 2_l \land 4_l \Rightarrow \exists_{l'} \gamma(l', l) = l \land 7_{(l,l')} \land 5_{l} \land 6_{l}, \text{ and} \)
6. \( (1_l \lor 4_l) \land (2_l \lor 3_l) \Rightarrow (5_l \Leftrightarrow 6_l) \)

The types of rules presented above give us a nice starting point in the development of syntactic conditions on SOS rules that guarantee associativity of operators while covering many practical applications. These rule types cover all rules that are in the de Simone format with four additional restrictions:
1. the target of the conclusion can contain at most one (binary) operator,
2. the aforementioned operator is the same as the one appearing in the source,
3. the labels of the premises and the conclusion coincide (apart from instances of the communicating rule), and
4. testing is disallowed, i.e., if a variable from the source of the conclusion appears in the source of a premise, the target of the same premise must appear in the target of the conclusion.

In [18], we present extensions of these rule types with various syntactic features such as testing and predicates.

**Theorem 12.** For a TSS in the ASSOC-de Simone format with respect to \( f \in \Sigma \), it holds that \( f \) is associative for each notion of equivalence \( \sim \) that includes strong bisimilarity.

Next, we present a few applications of the ASSOC-de Simone rule format to operators from the literature.

**Example 13** (Parallel composition). Consider the semantics of ACP parallel composition given in Example 8. Assume that the partial function \( \text{comm} \) on labels is also associative (as well as commutative). Thus, in terms of the types of deduction rules, we have deduction rules of type 1, 2, and 7. Therefore, the requirements in Definition 11 are met and it can be concluded from Theorem 12 that parallel composition is associative.

**Example 14** (Nondeterministic choice). Consider the semantics of nondeterministic choice presented in Example 9. In this TSS we find deduction rules of types 3 and 4. Therefore, the requirements of Definition 11 are met and it can be concluded from Theorem 12 that nondeterministic choice is associative.

**Example 15** (Disrupt). The disrupt operator was originally introduced in the language LOTOS [17], where it is used to model, for example, exception handling. This operator is also used, for instance, in [9], for the description of mode switches.

\[
\begin{align*}
\text{x}_0 & \xrightarrow{f} \text{y}_0 \\
\text{x}_0 & \triangleright \text{x}_1 \xrightarrow{f} \text{y}_0 \triangleright \text{x}_1 \\
\text{x}_0 & \triangleright \text{x}_1 \xrightarrow{f} \text{y}_1
\end{align*}
\]

Here we see that only deduction rules of types 1 and 4 are present. As a consequence of Theorem 12 also disrupt is associative.
5 Idempotence Format

Idempotence is a property of binary composition operators requiring that the composition of two identical specifications or programs will result in a piece of specification or program that is equivalent to the original components.

Definition 16 (Idempotence). A binary operator \( f \in \Sigma \) is idempotent w.r.t. an equivalence \( \sim \) if and only if the following equation is sound w.r.t. \( \sim \):

\[
f(x_0, x_0) = x_0.
\]

We now present a rule format guaranteeing the idempotence of certain binary operators. It is worth noting that the rule format described below relies on the determinism of certain transition relations. (We recall that a transition relation \( \rightarrow \) is deterministic when, for all closed terms \( p, p', p'' \), if \( p \rightarrow p' \) and \( p \rightarrow p'' \), then \( p' \equiv p'' \).) Determinism and idempotence may seem unrelated at first sight. However, it turns out that in order to obtain a powerful rule format for idempotence, we need to have the determinism of certain transition relations in place. Example 20 to follow witnesses the role that determinism plays in applications of our format to operations from the literature.

Definition 17 (The Idempotence Rule Format). Let \( \gamma : L \times L \rightarrow L \) be a partial function such that \( \gamma(l_0, l_1) \in \{l_0, l_1\} \) if it is defined. We define the following two rule forms.

1). Choice rules

\[
\begin{align*}
\frac{\{x_i \rightarrow t\} \cup \Phi}{f(x_0, x_1) \rightarrow t} & \quad i \in \{0, 1\}
\end{align*}
\]

2). Communication rules

\[
\begin{align*}
\frac{\{x_0 \rightarrow t_0, x_1 \rightarrow t_1\} \cup \Phi}{f(x_0, x_1) \rightarrow f(t_0, t_1)} & \quad l_0 \equiv t_1 \text{ or } (l_0 = l_1 \text{ and } \rightarrow \text{ is deterministic })
\end{align*}
\]

In each case, \( \Phi \) can be an arbitrary, possibly empty set of formulae.

In addition, we define the starred version of each form, 1\( ^* \) and 2\( ^* \).
Communication rules

\[
\begin{align*}
\{x_0 &\xrightarrow{l_0} y_0, x_1 \xrightarrow{l_1} y_1\} \\
\Rightarrow \quad f(x_0, x_1) &\xrightarrow{\gamma(l_0, l_1)} f(y_0, y_1)
\end{align*}
\]

A TSS is in idempotence format w.r.t. a binary operator \( f \) if

- each deduction rule is \( g \)-defining for some operator \( g \),
- each \( f \)-defining rule is of the forms \( 1_l \) or \( 2_{l_0,l_1} \), for some \( l, l_0, l_1 \in L \), and
- for each label \( l \in L \) there exists at least one rule of the forms \( 1^*_l \) or \( 2^*_{l_0,l_1} \).

In [3], we give syntactic criteria guaranteeing the determinism of certain transition relations. These syntactic constraints can be used to check the side condition on rules of the form \( 2_{l_0,l_1} \) and \( 2^*_{l_0,l_1} \).

**Theorem 18.** Assume that a TSS is in the idempotence format with respect to a binary operator \( f \). Then, \( f \) is idempotent w.r.t. to any equivalence \( \sim \) that includes strong bisimilarity.

**Example 19** (Non-deterministic choice). Consider again the semantics of non-deterministic choice presented in Example 9. Clearly, the rules given in that example are in the idempotence format w.r.t. \( + \). Hence, it follows from Theorem 18 that \( + \) is idempotent w.r.t. any equivalence that includes strong bisimilarity.

**Example 20** (Strong Time-Deterministic Choice). The choice operator that is used in the timed process algebra ATP [33] has the following deduction rules, where the special label \( \chi \) denotes the passage of one time unit.

\[
\begin{align*}
x_0 &\xrightarrow{a} y_0 \\
x_1 &\xrightarrow{a} y_1 \\
x_0 &\xrightarrow{\chi} y_0 \\
x_1 &\xrightarrow{\chi} y_1 \\
x_0 \oplus x_1 &\xrightarrow{a} y_0 \\
x_0 \oplus x_1 &\xrightarrow{a} y_1 \\
x_0 \oplus x_1 &\xrightarrow{\chi} y_0 \oplus y_1
\end{align*}
\]

The idempotence of this operator follows from our format since the last rule for \( \oplus \) fits the form \( 2^*_{\chi,a} \) because the transition relation \( \chi \) is deterministic over ATP.

## 6 Deriving Sound and Ground-Complete Axiomatizations

Sound and (ground-)complete axiomatizations are central notions to the algebraic treatment of programming and specification languages and, in particular, to process algebras [14, 26, 27]. They capture the basic intuition behind the algebra, and
The bulletin of the EATCS

the models of the algebra are expected to respect this intuition (e.g., the models induced by the operational semantics modulo bisimilarity). One of the benefits of having complete axiomatizations is that they enable reasoning at the level of syntax without committing to particular semantic models. When the semantic model of behaviour (e.g., the transition system associated to a term) is infinite, these syntactic techniques may come in very handy.

In [4], an automatic method for generating sound and ground-complete axiom systems for strong bisimilarity over the transition systems induced by GSOS language specifications is presented. It is assumed that there are only transition systems \[ \rightarrow \subseteq C(\Sigma) \times L \times C(\Sigma). \] In the remainder, we give a short overview of the technique presented in [4] and point out several variants and extensions thereof in the literature.

The approach of [4] is based on a restricted form of SOS deduction rules, called the GSOS format [16]. In what follows, for the sake of simplicity and uniformity, we only consider the positive subset of the GSOS format, defined below.

**Definition 21.** *(Positive GSOS Format)* A deduction rule is in the positive GSOS format when it is of the following form.

\[
\frac{\{x_i \xrightarrow{a_{ij}} y_{ij} \mid i \in I, j \in J_i\}}{f(\vec{x}) \xrightarrow{a} t}
\]

where

- the variables in \(\vec{x} = (x_0, \ldots, x_{n-1})\) and \(\{y_{ij} \mid i \in I, j \in J_i\}\) are all pairwise distinct,
- \(I\) is a subset of \(\{0, \ldots, n-1\}\), where \(n\) is the arity of \(f\),
- \(J_i\) is a finite index set, for each \(i \in I\), and
- \(\text{vars}(t) \subseteq \{x_0, \ldots, x_{n-1}\} \cup \{y_{ij} \mid i \in I, j \in J_i\}\).

It is well-known that some operators whose operational semantics can be expressed in the positive GSOS format cannot be finitely axiomatized modulo bisimilarity. Therefore, in order to axiomatize them finitely, one needs auxiliary operators. (See, e.g., [29].) Thus, in order to achieve a finite ground-complete axiomatization, we may need to extend the signature of a language with fresh operators. The kind of extension required for this purpose is called disjoint extension and is defined below.

\[ 93 \]
**Definition 22 (Disjoint Extension).** Consider TSSs $T_1 = (\Sigma_1, L_1, D_1)$ and $T_2 = (\Sigma_2, L_2, D_2)$. TSS $T_2$ is a disjoint extension of TSS $T_1$ if and only if $\Sigma_1 \subseteq \Sigma_2$, $D_1 \subseteq D_2$ and the operators from $\Sigma_1$ do not occur in the sources of the deduction rules from $D_2 \setminus D_1$.

The crucial property of the notion of disjoint extension that underlies the developments to follow is that if $T_2$ is a disjoint extension of $T_1$, then two closed terms over $\Sigma_1$ are strongly bisimilar w.r.t. $T_1$ if and only if they are strongly bisimilar w.r.t. $T_2$. This means that an axiom system that is sound and ground-complete w.r.t. bisimilarity over $T_2$ can be used to show all the valid equalities between closed terms over the signature of $T_1$.

Next, we define when a TSS does not allow infinite traces. Such a TSS is called *trace finite*. In [4], syntactic criteria are given for guaranteeing trace finiteness of a TSS.

**Definition 23 (Trace Finiteness).** Let $T$ be a TSS in the positive GSOS format. A term $p \in C(\Sigma)$ is trace finite if there exists no infinite sequence $p_0, l_0, p_1, l_1, \ldots$ of closed terms $p_i$ and labels $l_i$ such that $p \equiv p_0$ and $p_i \xrightarrow{l_i} p_{i+1}$, for all $i \geq 0$. The TSS $T$ is trace finite iff all terms in $C(\Sigma)$ are trace finite.

The following theorem from [4] states that for trace-finite TSSs in the positive GSOS format, we can always add sufficiently many auxiliary operators in order to generate a finite ground-complete axiomatization of strong bisimilarity. The procedure to obtain such a ground-complete axiomatization is sketched subsequently.

**Theorem 24.** Let $T$ be a trace-finite TSS in the GSOS format. Then, there are a disjoint extension $T'$ of $T$ and a finite axiom system $E'$ such that $E'$ is a sound and ground-complete axiomatization of bisimilarity on closed terms from $T'$.

The approach of [4] relies on the presence of three basic operators in the signature, namely a constant $0$, denoting inaction or deadlock, a unary action prefixing operator $a._x.$ for each $a \in L$, and a binary nondeterministic choice $._+.$, already presented in Example 9. These operators allow one to denote all finite synchronization trees in the sense of Milner [27].

The inaction constant $0$ has no transition and hence it has no defining rule in the semantics. The deduction rule for action prefixing is given below and the deduction rules for nondeterministic choice are those presented in Example 9.

\[
\begin{align*}
& \text{a}_x_0 \xrightarrow{a} x_0 \\
& \text{These basic operators are finitely axiomatized modulo bisimilarity by the following axiom system [25].}
\end{align*}
\]
The Bulletin of the EATCS

\[
\begin{align*}
  x_0 + x_1 &= x_1 + x_0 \\
  (x_0 + x_1) + x_2 &= x_0 + (x_1 + x_2) \\
  x_0 + x_0 &= x_0 \\
  x_0 + 0 &= x_0
\end{align*}
\]

Note that the above axioms are sound not only for the small TSS containing only the three operators introduced above, but also for any disjoint extension thereof. Hence, if the TSS to be axiomatized does not contain these operators, we can safely add them to the signature and the above-given axiom system remains sound in this extended TSS as well as in all of its disjoint extensions. All the equations that are generated by the method in [4] are “robust”, in the sense that they remain valid for any disjoint extension of a GSOS language.

The idea behind the procedure for the automatic generation of finite axiomatizations of bisimilarity presented in [4] is as follows. Assume that we have a TSS \( T \) in (positive) GSOS format that disjointly extends the TSS described above for finite synchronization trees. Since we already have the above-given complete axiomatization of bisimilarity over finite synchronization trees, in order to obtain a ground-complete axiomatization of bisimilarity over \( T \), it suffices only to generate a disjoint extension of \( T \) and a finite axiom system that can be used to rewrite each closed term into an equivalent finite synchronization tree.

The axiomatization procedure starts with the axiom system given above. Then, on top of these laws, for each operator we generate new axioms by using the above-mentioned operators and other auxiliary ones. Some operators, namely the smooth and distinctive ones introduced in Definition 26 to follow, distribute w.r.t. nondeterministic choice in some of their arguments and can be handled without recourse to auxiliary operators. Other operators, namely the smooth ones that are not distinctive, can be expressed as nondeterministic compositions of auxiliary operators that are smooth and distinctive. Others still are expressed in terms of auxiliary smooth operators with possibly different arity.

**Definition 25.** (Smooth Operators) An \( n \)-ary function symbol \( f \) is called smooth when all \( f \)-defining deduction rules are of the following form

\[
\frac{\{ x_i \xrightarrow{a} y_i \mid i \in I \} }{ f(x_0, \ldots, x_{n-1}) \xrightarrow{a} C[\overrightarrow{x}, \overrightarrow{y}] },
\]

where \( I \subseteq \{0, \ldots, n-1\} \), \( C[\overrightarrow{x}, \overrightarrow{y}] \) is a term containing only variables in \( \overrightarrow{x} \) and \( \overrightarrow{y} \) and moreover no \( x_i \) appearing in the source of a premise appears in the target of the conclusion.
Definition 26. (Distinctive Operators) A smooth operator is called distinctive, when for each two distinct \( f \)-defining rules of the following form

\[
\begin{align*}
\{ x_i \xrightarrow{a_i} y_i \mid i \in I \} \\
\{ x'_i \xrightarrow{a'_i} y'_i \mid i \in I' \}
\end{align*}
\]

\[
\begin{align*}
f(x_0, \ldots, x_{n-1}) \xrightarrow{a} C[x, y] \\
f(x'_0, \ldots, x'_{n-1}) \xrightarrow{a'} C'[x', y']
\end{align*}
\]

it holds that \( I = I' \) and there exists an \( i \in I \) such that \( a_i \neq a'_i \).

Example 27. The parallel composition operator of Example 8 is smooth but not distinctive.

To axiomatize a distinctive operator \( f \), it suffices to add the following equations, which describe the interplay between \( f \) and the operations for finite synchronization trees, to our axiom system.

1. Distributivity laws: If \( i \in I \) for each deduction rule of the form given in Definition 25, we have the equation:

\[
f(x_0, \ldots, x_i + x'_i, \ldots, x_{n-1}) = f(x_0, \ldots, x_i, \ldots, x_{n-1}) + f(x'_0, \ldots, x'_i, \ldots, x_{n-1}).
\]

2. Action laws: For each \( f \)-defining rule of the form given in Definition 25, we have the equation

\[
f(P_0, \ldots, P_{n-1}) = a.C[P, y],
\]

where \( P_i = a_i.y_i \), if \( i \in I \), and \( P_i = x_i \), otherwise.

3. Inaction laws: We have a law of the form

\[
f(P_0, \ldots, P_{n-1}) = 0,
\]

where

- each \( P_i \) is of the form \( 0, x_i \) or \( b_i.x_i \) for some label \( b_i \) and
- for each deduction rule of the form given in Definition 25, there is an index \( i \in I \) such that \( P_i = 0 \) or \( P_i = b_i.x_i \), for some \( b_i \neq a_i \).

The axiom system obtained thus far gives a sound and ground-complete axiomatization of smooth and distinctive operators (possibly over a disjoint extension of the original language with inaction, action prefixing and nondeterministic choice). In order to axiomatize smooth operators that are not distinctive, we partition their deduction rules into sets of rules satisfying the criteria of Definition 26. This is always possible and in the worst case singleton sets of rules trivially
satisfy these criteria. Then, for each partition, we introduce an auxiliary operator with the defining rules given in that partition. The introduced operators are by construction smooth and distinctive. Hence, we can apply the construction given before to these operators to axiomatize them.

Assume that for a smooth but non-distinctive operator $f$, the set of deduction rules is partitioned into $n$ sets and thus $n$ smooth and distinctive function symbols $f_0$ to $f_{n-1}$ are introduced. Then, in addition to the axiom system for each auxiliary operator, we introduce the following axiom.

$$f(x_0, \ldots, x_{n-1}) = f_0(x_0, \ldots, x_{n-1}) + \cdots + f_{n-1}(x_0, \ldots, x_{n-1})$$

The above axiom together with the axioms generated for $f_0, \ldots, f_{n-1}$ give a sound and ground-complete axiomatization for the smooth operator $f$.

The following example illustrates this procedure.

**Example 28.** Consider again the parallel composition operator of Example 8. The trivial partitioning of its deduction rules gives rise to 3 auxiliary operators $\parallel_0$, $\parallel_1$, and $\parallel_2$, with the following semantics.

\[
\begin{align*}
(p0) & \quad x_0 \xrightarrow{f} y_0 & \quad x_0 \parallel_0 x_1 \xrightarrow{f} y_0 \parallel_1 x_1 \\
(p1) & \quad x_1 \xrightarrow{f} y_1 & \quad x_0 \parallel_1 x_1 \xrightarrow{f} x_0 \parallel_1 y_1 \\
(p2) & \quad x_0 \xrightarrow{f} y_0 & \quad x_0 \parallel_2 x_1 \xrightarrow{f} y_0 \parallel_2 y_1
\end{align*}
\]

Then, applying the procedure given before, we obtain the following axiom system.

\[
\begin{align*}
x_0 + x_1 & = x_1 + x_0 & (x_0 + x_2) \xrightarrow{f} x_2 & = x_0 + (x_1 + x_2) \\
x_0 + x_0 & = x_0 & x_0 + 0 & = x_0 \\
(a.x_0) \parallel_0 x_1 & = a.(x_0 \parallel_1 x_1) & x_0 \parallel_1 (a.x_1) & = a.(x_0 \parallel_1 x_1) \\
(a.x_0) \parallel_2 (b.x_1) & = c.(x_0 \parallel_2 x_1) & \text{if } \text{comm}(a, b) = c \\
0 \parallel_0 x_1 & = 0 & x_0 \parallel_0 0 & = 0 \\
0 \parallel_2 x_1 & = 0 & x_0 \parallel_2 0 & = 0 \\
(a.x_0) \parallel_2 (b.x_1) & = 0 & \text{if } \text{comm}(a, b) \text{ is undefined} \\
(x_0 + x_1'') \parallel_0 x_1 & = (x_0 \parallel_0 x_1) + (x_1'' \parallel_0 x_1) \\
x_0 \parallel_1 (x_1 + x_1') & = (x_0 \parallel_1 x_1) + (x_0 \parallel_1 x_1') \\
(x_0 + x_1'') \parallel_2 x_1 & = (x_0 \parallel_2 x_1) + (x_1'' \parallel_2 x_1) \\
x_0 \parallel_2 (x_1 + x_1') & = (x_0 \parallel_2 x_1) + (x_0 \parallel_2 x_1') \\
x_0 \parallel_1 x_1 & = x_0 \parallel_0 x_1 + x_0 \parallel_1 x_1 + x_0 \parallel_2 x_1
\end{align*}
\]

The generated axioms do resemble the original axioms of [14] to a large extent. The auxiliary operators $\parallel_0$, $\parallel_1$, and $\parallel_2$ are called left, right and communication merge in the literature.
Using the techniques introduced so far, one can axiomatize smooth operators. Non-smooth operators may test some of their arguments or make copies of them. As described in detail in [4, Section 4], we can axiomatize an $n$-ary non-smooth operator $f$ by means of an $m$-ary smooth operator $g$ that simulates all the copying and testing done by the rules for $f$. The following example illustrates this, rather technical, procedure on a simple non-smooth operation.

**Example 29.** Consider a (hypothetical) unary operator $f$ with the following operational semantics.

\[
\begin{align*}
x_0 & \rightarrow_{a} y_0 \\
x_0 & \rightarrow_{b} y'_0 \\
f(x_0) & \rightarrow_{a} g(x_0, y'_0)
\end{align*}
\]

The function symbol $f$ is not smooth, because $x_0$ appears twice as the source of premises and, moreover, both $x_0$ and the result of one of its transitions, i.e., $y'_0$, appear in the target. To remedy this, we introduce a binary auxiliary operator $h$ with the following semantics.

\[
\begin{align*}
x_0 & \rightarrow_{a} y_0 \\
x_1 & \rightarrow_{b} y_1 \\
h(x_0, x_1) & \rightarrow_{a} g(x_0, y_1)
\end{align*}
\]

The function symbol $h$ is now smooth and thus, the procedure given before can readily axiomatize it. Adding the following equation will complete the axiomatization of $f$:

\[f(x_0) = h(x_0, x_0).\]

To conclude, using the procedure sketched above, by adding sufficiently many auxiliary operators, one can finitely axiomatize bisimilarity over closed terms in any TSS in the positive GSOS format.

A generalization of Theorem 24 to non-trace-finite and non-positive TSSs is also presented in [4]; the generalization to non-trace-finite TSSs requires the addition of an infinitary conditional equation, the *Approximation Induction Principle* from [10], to the generated axiom system.

The techniques from [4] were extended in [13] to cater for explicit termination of processes. This approach, although more complicated in nature, gives rise to more intuitive and more compact sets of equations compared to the original approach of [4]. The resulting format is called the TAGH format. (The acronym TAGH format stands for *termination and GSOS hybrid* format.)

The definition of languages in the GSOS and the TAGH formats requires that the signature, the set of action labels, and the set of deduction rules be finite. In [1], Aceto defines the infinitary GSOS format. It extends the GSOS format by allowing for a countable signature, a countable set of action labels, and a countable
set of deduction rules. The sub-format regular GSOS guarantees that every closed
term describes a finite labelled transition system [2]. For this sub-format, Aceto
presents a variation on the procedure described above that allows one to generate
a sound and ground-complete axiomatization for bisimilarity, which makes use of
the Recursive Specification Principle [12].

Bloom [15] has shown that the approach of [4] can also be used for gener-
ating axioms for rooted branching bisimilarity, and van Glabbeek claims in [22]
that this is also the case for rooted-\(\eta\) bisimilarity. The latter work also offers an
adaptation of the approach of [4] that yields finite, sound and ground-complete
axiomatizations for rooted branching and rooted delay bisimilarities [20].

Axiom systems for preorders have been generated, too, see for instance [39].
Along the same lines, [40] generates prioritized rewrite systems for TSSs with an
ordering on deduction rules (see [30]).

7 Conclusions and Open Problems

In this paper, we have presented an overview of some of the existing meta-results
for guaranteeing the validity of algebraic properties and for generating ground-
complete axiom systems from operational semantics. There is an ongoing re-
search in this field and we currently have proposals for rule formats for unit and
zero elements; see [8]. A closely related line of research aims at developing (pos-
sibly automatic) proof techniques for establishing the soundness of axiom systems
using the SOS specification of a language; see, e.g., the papers [19, 42, 5].

There remain many open problems to be addressed. An extension of the
ASSOC-de Simone format with negative premises seems a challenging research
problem to us. Also, relaxing the formats of Sections 3-5 to guarantee algebraic
properties for weaker notions of behavioural equivalence than bisimilarity could
be worth investigating. If a TSS conforms to any of the rule formats presented in
this paper, any disjoint extension of such a TSS also conforms to the same format.
Hence, the algebraic properties proven by the meta-theorems are robust, in
the sense that they remain sound under any disjoint extension. It is in general very
challenging and interesting to address the robustness of algebraic properties under
extensions of TSSs. Another interesting topic for future research is to combine the
techniques of Sections 3-5 with those presented in Section 6 in order to generate
“more natural” axiomatizations, which resemble the axiomatizations presented so
far by the language designers. A challenging open problem in this research area
is the development of methods for the automatic generation of axiomatizations of
behavioural equivalences that are complete over arbitrary open terms. The hard-
ness of this problem is witnessed by the lack of such results even for specific
process algebras that contain operators like restriction and parallel composition.
with synchronization. See [6] for a survey of results on complete axiomatizations up to 2005.

References


The Bulletin of the EATCS


The Bulletin of the EATCS


**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 of Computer Science (ICS)
Foundation for Research and Technology (FORTH)
N. Plastira 100. Vassilika Vouton
GR-700 13 Heraklion, Crete, Greece
faturu@csd.uoc.gr

**Composable Error Recovery With Transactional Memory**

Torvald Riegel
Technical University Dresden (TUD)
torvald.riegel@tu-dresden.de

Pascal Felber
University of Neuchatel
pascal.felber@unine.ch

Christof Fetzer
Technical University Dresden (TUD)
christof.fetzer@tu-dresden.de

Abstract

Implementing correct error handling is a difficult task. Even though modern computing languages provide sophisticated mechanisms for runtime error handling, notably exceptions, recovering from errors during execution...
remains far from trivial. The major sources of complexity include rolling back the partial effects of a failed execution, the coordination necessary between the components that fail and the code that handles the error, and the lack of composability of error handling mechanisms. In this paper, we propose to combine error handling mechanisms based on atomic blocks, implemented using transactional memory, and a recovery manager that automates and coordinates the recovery process. Error recovery can be handled locally only in simple scenarios. In complex systems, coordination is necessary during recovery to handle the dependencies between the objects and their respective environment. We show how a recovery manager can have objects cooperate in the recovery process and, with the help of atomic blocks, make error recovery composable.

1 Introduction

Robust software has to remain operational despite the occurrence of errors and, therefore, must be able to tolerate runtime failures. Ideally, error handling and recovery should not introduce a substantial increase in the code complexity, as it might increase the probability of design and coding faults and thus decrease the robustness of the application.

Modern programming languages implement runtime error handling using exceptions, a powerful mechanism that separates error handling code from functional code and allows a clean path for error propagation. When a semantic constraint is violated or when some exceptional error condition occurs, an exception is thrown. This causes a non-local transfer of control from the point where the exception occurred to a point, specified by the programmer, where the exception is caught. An exception that is not caught in a method is implicitly propagated to the calling method.

In contrast, in programming languages without exceptions such as C, the programmer needs to check for error return codes after each function call. This tends to produce intricate error handling code interleaved with functional code, which in turn makes recovery complex and error-prone. The use of exception handling mechanisms can thus simplify the development of robust programs.

When an application encounters an exceptional situation, it can either try to recover from the error and proceed (resumption semantics), or give up and propagate the error to be handled by the caller (termination semantics). For a recovery to be successful, one must first guarantee that the premature exit of a method, caused by an exception, does not leave an object in an inconsistent state. This can be achieved by undoing the partial effects of actions that have failed (roll back). Second, the application must be able to fix the runtime error condition (repair) and execute again the failed operations (retry), or continue its processing by following
Despite their many benefits, exceptions are no panacea. Their apparent simplicity might lead to the neglect of recovery issues [9] and produce buggy code. Furthermore, they complicate the application’s control flow without significantly decreasing the size of error handling code. Often more than two thirds of the code [11], and three quarters of carrier-grade code [31], is devoted to detecting and handling errors and exceptions. Exception handling code is also more likely to contain software bugs than other parts of an application, because it is rarely exercised and hence not well tested.

Another important limitation is that recovery code implemented using exceptions is typically not composable: one cannot recover from an error occurring during a set of operations by simply composing the recovery actions of individual operations. Consider method swapTop() in Figure 1 that exchanges the two topmost elements of a stack. Even if the individual pop() and push() operations correctly handle errors and leave the stack in a consistent state when throwing an exception, the failure of, say, the second call to pop() will break atomicity of the swap operation. Therefore, recovery code is necessary to either undo partial execution, or complete the operation.

Figure 2 illustrates the lack of composability of recovery code based on exception handling. To preserve failure atomicity, i.e., atomic behavior in the case of a failure, one needs to keep track of which operations have been completed before the exception and only undo those (e.g., by executing compensation actions). One can observe that implementing correct recovery code using exceptions is non-trivial. Consider that the compensation actions on lines 17, 19, and 21 might themselves throw exceptions and prevent recovery. A better implementation is shown in Figure 3, where exceptions in compensations are caught and state corruption is signaled to the caller.

In summary, exception handling suffers from two major problems that hinder error recovery. First, they do not provide failure atomicity: rollback has to be performed explicitly and is often complex. Second, recovery is not composable: when a computation is composed of different operations, the recovery strategies

```
1 class Stack {
2     ...
3     void swapTop() { // Swap topmost elements
4         Object item1, items;
5         item1 = pop(); // Might throw exception
6         item2 = pop(); // Might throw exception
7         push(item2); // Might throw exception
8         push(item1); // Might throw exception
9     }
10 }
```

Figure 1: Exchanging the topmost objects of a stack.
class Stack {
    
    void swapTop() { // Swap topmost elements
        Object item1, items;
        int step = 0;
        try {
            item1 = pop(); // Might throw exception
            step = 1;
            item2 = pop(); // Might throw exception
            step = 2;
            push(item2); // Might throw exception
            step = 3;
            push(item1); // Might throw exception
        } catch (Exception e) {
            switch (step) { // Roll back changes
                case 3:
                    pop(); // Pop item2
                case 2:
                    push(item2); // Push back item2
                case 1:
                    push(item1); // Push back item1
                }
            throw new AbortedException(e); // Indicate failure
        }
    }
    ...

Figure 2: Stack element swap with naive error recovery.

for each operation need to be coordinated, which, among other problems, breaks
the information hiding principle.

In this paper, we propose an original approach to providing composable error
recovery. It combines atomic blocks, implemented using transactional memory,
as an alternative to exception handling for implementing roll-back upon failure
with a recovery manager that automates and coordinates the recovery process and
provides generic error recovery strategies.

The rest of the paper is organized as follows: We first give an overview of
related work in Section 1.1 and then highlight how atomicity helps in Section 1.2.
We analyze the problem of composable error recovery and explain our approach
on a conceptual level in Section 2. We present a programming language interface
for recovery in Section 3 and the concept of a recovery manager in Section 4,
evaluate our approach in Section 5, and conclude in Section 6.

1.1 Background and Related Work

Exception handling has been investigated for several decades. Goodenough [15]
proposed to add explicit programming language constructs for exception handling
in 1975, and Melliar-Smith and Randell [24] introduced the combination of recovery
blocks [5] and exceptions to improve the error handling of programs in 1977.
Exception handling is still actively investigated (e.g., [2], [29]). One of the ma-
The Bulletin of the EATCS

```java
11 class Stack {
12    ...
13    void swapTop() { // Swap topmost elements
14        Object item1, items;
15        int step = 0;
16        try {
17            item1 = pop(); // Might throw exception
18            step = 1;
19            item2 = pop(); // Might throw exception
20            step = 2;
21            push(item2); // Might throw exception
22            step = 3;
23            push(item1); // Might throw exception
24        } catch (Exception e) {
25            try {
26                switch (step) { // Roll back changes
27                    case 3:
28                        pop(); // Pop item2
29                    case 2:
30                        push(item2); // Push back item2
31                    case 1:
32                        push(item1); // Push back item1
33                }
34            } catch (Exception e2) { // Ooops! What can we do?
35                throw new IllegalStateException(); // Stack is now useless
36            }
37        }
38    } catch (Exception e) { // Indicate failure
39    }
40    ...
41 }
```

Figure 3: Stack element swap with improved error recovery.

Major issues addressed by researchers is a better separation of functional code and exception handling code. Recent studies have proposed to combine exception handling and reflection to increase this division [25], or to use aspect-oriented programming for reducing the amount of code related to exception handling [21].

To verify the correctness and failure atomicity of exception handling code, one can use static analysis [33], dynamic testing tools based on exception injection [14], dependability cases [23], or various other testing techniques.

Atomicity can also be guaranteed by using transactional mechanisms that automate state roll-back upon failure. Transactional memory (e.g., [17, 20, 26]) is a means to execute application code that accesses main memory transactionally (i.e., virtually isolated from other transactions and atomically). It provides the underlying mechanisms for undoing partial effects of a failed computation. Harris et al. have proposed language support for lightweight transactions in the form of “atomic blocks” [17, 19]. A retry statement can be used to force the atomic block to be retried. Alternatives can be specified with the help of do/or else blocks. The execution of a retry within the do block will lead to the execution of the or else block before the enclosing atomic block is retried. See Section 5.1 for a detailed comparison with our approach. Current software implementations
of transactional memory have moderate single-thread overheads [10, 30].

Various approaches have been proposed for implementing recovery. The basic principle of reattempting a failed computation after a failure has been used for decades [16].

Xept [32] is a language and tool to handle exceptions from function calls. It works by intercepting selected functions and performing both error detection and error recovery. Recovery-oriented computing (ROC) [27] takes the perspective that errors are facts to be coped with, not problems to be solved. The entire software is restructured to allow recovery, by isolating components and making them failure aware. Micro-reboots [8] have been proposed as a way to rejuvenate components even before they experience a failure. Rx [28] rolls back the program to a previous checkpoint upon failure and modifies the environment before re-executing the computation. We follow a different approach in that we rely upon a recovery manager to automate failure handling and recovery. The developer can specify alternative execution paths and rollback is implemented using language-level atomic blocks. Note that the recovery manager can use any available mechanism, such as micro-reboots or environment modifications, to increase the likelihood of success upon retry.

The approach presented in this paper is a continuation of our previous work [13]. We present a detailed comparison with related proposals in Section 5.1, after describing our approach in the following sections.

1.2 Failure Atomicity

"... the fundamental premise of transactions is that we needed exception handling in distributed computations. Transactions are the computer equivalent of contract law. If anything goes wrong, we’ll just blow away the whole computation. Absent some better model, that’s a real simple model that everybody can understand." — Jim Gray, 2003 [1].

If one has access to transactions at the programming level, rolling back partial effects becomes much simpler (as pointed out by [14, 18]): if an error occurs, the transaction simply needs to be aborted and modifications to main memory are undone. This effectively ensures failure atomicity. Furthermore, transactions are guaranteed to be executed in isolation and cannot see the modifications performed by uncommitted concurrent transactions.

Transactions are composable. Small atomic actions can be composed into larger actions that still execute atomically. Implementing error recovery in the stack element swap example is as simple as putting the individual operations in
The Bulletin of the EATCS

class Stack {
    _
    void swapTop() { // Swap topmost elements
        atomic {
            Object item1, items;
            item1 = pop();
            item2 = pop();
            push(item2);
            push(item1);
        }
    }
}

Figure 4: Stack element swap using atomic blocks.

an atomic block (see Figure 4). In turn, the pop() and push() operations may contain atomic blocks internally, effectively producing nested transactions [12].

Transactional memory (see Section 1.1) provides transactions for main memory. It is easy to use by developers as memory accesses are transparently delegated to the transactional memory (either using compiler or hardware support). Transactional memory makes concurrent programming much easier, and we expect it to become widely available as multi-core CPUs and multiprocessor systems become ubiquitous.

If an error is encountered during the execution of an atomic block, one would intuitively assume that the underlying transaction is aborted and possibly retried. However, the decision whether to retry an aborted transaction or not is already a recovery decision. For example, how often should we retry after errors? This decision is something that a transaction cannot reliably specify on its own (see Section 2). It most likely needs runtime information to make a good decision.

Atomic blocks, and atomicity in general, are very helpful to ensure consistency and to roll back incomplete actions. However, this is only one part of error recovery. For the second part, the actual recovery strategy, another mechanism is needed, which we will describe in the next sections.

2 Problem Statement and Approach

In this section, we explain why using atomic blocks is necessary but not sufficient in order to achieve composability for error recovery. We propose the concept of a recovery manager that controls and coordinates the recovery strategies of the individual components of a system. We first motivate this by explaining why coordination and generic support for recovery is required, and then describe our approach on a conceptual level.
2.1 Conceptual Reasons

First, error handling and recovery is a cross-cutting concern. Exceptions can happen at many places in an application. Exceptions are either truly exceptional in the sense that they are not considered to belong to the core application logic, or they are used as a more convenient way to return results.

For truly exceptional errors (e.g., programming errors, bit flips, memory exhaustion), there is typically no special way to handle them that is strongly related to the place in the application where they occurred. Thus, a generic way to handle them would be beneficial, even for applications that have only very coarse error handling.

For exceptions used in the application logic to convey results, we need to distinguish between different types of applications. Applications with no or very coarse-grained exception handling would benefit from additional, generic exception handling capabilities that can be plugged in. When applications have more elaborate exception handling, they typically contain expert knowledge in the form of precise reactions to certain errors. However, these reactions can easily get repetitive and error-prone, similar to traditional return code checking. Thus, even for these applications, we would like to promote simple and generic mechanisms that would avoid reimplementing the same error handling scheme at several places in the application.

Faults that activate an error can also be located in other components that are not directly related to the component with the error, for example, if both components share resources (e.g., the application heap) or interact with the same objects. It is typically not known at design time that such interferences exist (e.g., with which other components resources are shared).

From an economic perspective, better support for recovery as a cross-cutting concern can make error handling simpler and decrease the development effort. Applications for which advanced error handling is deemed too costly in terms of development efforts can benefit from pluggable, generic recovery strategies.

Second, recovery cannot be handled locally in all cases. Sometimes, parts of an application need to cooperate with other parts, even if these other parts do not encounter the error. For example, objects share the application heap, so out-of-memory errors might be fatal for some objects, whereas other objects in the application could easily reduce their resource needs (e.g., an object that caches data for performance but can recompute it on demand). Similarly, some programming errors like race conditions can be handled by pausing other conflicting threads. When several objects provide alternative implementations, only the combination of some of these alternatives might result in an error-free execution. If this coordination would be directly implemented in every object or component, these parts would hardly be composable. Furthermore, the development effort would be very
Third, some recovery strategies require information only available at runtime or after deployment of the application. Relevant information includes which recovery strategies are provided by the target system, which other application parts might interfere, and issues specific to the environment such as hardware faults. Having to depend on such information at development time would hinder composability. Furthermore, the performance of recovery strategies can, for example, depend on how much memory versus how much computing power is available on the target system.

2.2 Our Approach

We use atomic blocks as foundation. They provide failure atomicity and allow a recovery strategy to try out different ways to reach success. Programmers can compose atomic actions from a single atomic block or from several atomic blocks, which allows them to express functionally equivalent alternatives as well as alternatives with degraded functionality. Additionally, we introduce repair actions as a means for objects to specify how they can potentially help if an error occurs in some other part of the system.

Alternatives and repair actions express options for how to perform a certain function. However, they do not specify a recovery or execution strategy because this cannot be decided locally in every case. Instead, we delegate this task to a module that we call the recovery manager (RM), a concept which we initially proposed in [13]. The RM controls which alternatives and repair actions are executed after an error occurred. It thus implements the recovery strategy, tries to ensure forward progress, and can coordinate recovery attempts of different components. Additionally to pure error handling, it can also perform fault handling [3] by proactively controlling execution to try to avoid faults. RMs can base their decisions on different kinds of input data, for example, information about errors that occurred, runtime measurements (e.g., heap usage), and statistics about previous behavior of programs. They also provide generic error handlers and cross-cutting error-related facilities such as logging and tracing.

Errors can be signaled by application code or by the RM (e.g., if it includes a facility to detect inconsistent state). Application code can either provide no additional information for an error, hints about the nature of the error (symptoms), or custom information for application-logic–specific (error contexts).

To ease error handling for programmers, we provide two composable mechanisms for signaling and handling errors. Implicit error handling is targeted at errors that are not a conceptual part of the application logic or for which no custom handling is wanted; programmers do not need to provide a custom error de-
scription and completely rely on the RM to recover from the error by exploring alternatives, generic error handlers, and other facilities (e.g., implicit logging and user notification). Explicit error handling can be used if errors really require application-specific handling that cannot be expressed through alternatives or with the help of generic handlers provided by the RM. Even with explicit error handling, the RM can still try to first execute generic recovery strategies before it switches to custom handlers.

Traditional exception handling mechanisms do not allow programmers to safely and efficiently implement or use these concepts. Exception handling is purely reactive and constrained to the control flow of the error. It does not provide an elegant way to express alternatives or to cooperate with other components. Both the try/catch exception handling interface and explicit return code checking express concrete recovery strategies because they imply fixed reactions to errors. This also makes error descriptions a critical property and results in difficult trade-offs [22] between precision of error descriptions and information hiding.

We therefore propose a modified programming language interface for error recovery in Section 3. A description of the recovery manager follows in Section 4.

3 Language Integration

In this section, we discuss design and semantics of the programming language interface for recovery. The goal for this interface is to closely match the intuition developers might have about error handling and to follow the approach described in Section 2. Where possible, we let the recovery manager (RM) provide features as ordinary methods or functions. We use the Java programming language as an example to illustrate the interface.

We first describe the language interface for implicit error handling and then for explicit error handling. Finally, we describe repair actions and how to integrate traditional exception handling.

3.1 Implicit Error Handling

We think that most of the time, the main concern of programmers is making the program do something by simply following the algorithm. Errors are a deviation from the planned route and thus increase complexity and difficulty. Errors will happen eventually but handling them explicitly is just a distraction.

We model the language interface accordingly. The basic interface that we propose consists of three elements: (1) atomic blocks demarcate groups of statements that must execute with “all-or-nothing” semantics, (2) atomic blocks can fail by
signaling that they encountered an error, and (3) atomic actions can consist of several alternative blocks that mark alternate routes to success.

The recovery manager (RM) controls how often atomic actions are retried and which alternatives are executed, it requests cooperation if necessary, it can interact with the application’s environment, and it will interact with users if necessary (e.g., report errors with a user-controllable level of detail). Programmers can give hints to the RM regarding errors, but this is not required.

Informally, this approach then allows programmers to just follow the route descriptions (i.e., the algorithm). If they get lost, they just say so (i.e., they fail) and will then be reset to a previous position. If the route parts (alternatives), they get told which alternative to choose. They just need to walk along routes, but they do not need to have and understand a complete map.

Examples for implicit error handling include functions that are required to succeed and for which every persistent error is fatal, functions that do not have application-specific errors, or functions that do not (want to) provide application-specific error handling code.

3.1.1 Atomic Actions and Atomic Blocks

An atomic action consists of at least one atomic block. We will later extend the notion of atomic actions to possibly contain several atomic blocks. The example of Figure 5 contains two atomic actions, each having a single atomic block. All the statements inside an atomic block are treated as a single transaction (see Figure 5). We use transactional memory (TM) to implement atomic blocks.

Atomic actions can be nested inside the atomic blocks of other atomic actions. We assume that the TM provides closed-nesting semantics [12]. Flat-transaction semantics are also possible but can possibly decrease performance. We call an atomic action that is not nested in another atomic block an “outermost atomic” action.

3.1.2 Signaling Errors

Errors (and faults) are either detectable by application logic (e.g., invalid input data led to state that cannot be handled by an algorithm) or not (e.g., bit flips).
Next, we describe how programs can signal the former. We assume that the latter will be detected by the RM and signaled on behalf of the thread executing an atomic action.

Because we want to provide implicit error handling, we do not require the programmer to specify the kind of the error or which fault caused it. Often, the programmer will not be able to easily obtain the real cause of an error, or he/she can only state a local, implementation-dependent view that might not be useful in the caller’s context.

The underlying motivation for making the error description optional is that programmers should be able to concentrate on the implementation of the object, and nothing else. The basic intuition of an error is that something is wrong, and it is acceptable for a programmer not to know what is wrong but to just signal the error. This is arguably better than having to pick some (possibly wrong) reason, which might then be misused by the recovery strategy.

For these reason, we propose only a very basic mechanism for propagating errors. A programmer can use the keyword fail to state that the program encountered an error, and execution of the atomic block that encountered the error cannot be continued. fail is implemented as an invocation of the RM, which then takes appropriate actions (e.g., aborting the transaction for the enclosing atomic block and retrying the execution). When signaling an error, the program only expresses that the current invocation of this block failed, but subsequent executions of the same block might still succeed. If an atomic action with a single atomic block fails, the atomic action fails. If a nested atomic action with a single atomic block fails, the error is signaled in the enclosing atomic block, so errors potentially propagate to the outermost atomic block (see Figure 6).

To provide more information such as local observations about the error, the program can use a fail statement with an attached atomic block (see second
fail in Figure 6). If the block itself fails, then this statement is equivalent to a fail without a block. If the block does not fail, it still will be rolled back and will inject a failure into the enclosing atomic block. However, the RM is allowed to use all the information that it has received during a successful execution of the block (Section 4.2 gives more examples for these hints). This mechanism thus allows the programmer to provide an error description in a safe and atomic way.

3.1.3 Alternatives for Atomic Actions

Programmers can provide alternative implementations of atomic actions by providing alternative atomic blocks. These are marked as blocks preceded by the keywords or and or else. Keyword or defines alternatives that are functionally equivalent and or else defines alternatives in which the alternatives before the keyword or else are preferred to the ones that follow it.

Figure 7 shows an example with two atomic actions each having two atomic blocks. The first alternative of each atomic action always fails, whereas, the second alternative of the nested action will not. Thus, the second alternative of the outermost action can also succeed. In this case, all three alternatives are equivalent from a functional point of view, i.e., the RM is free to try the three alternatives in any order. In particular, if the RM learns that the last alternative has the highest probability of success, it is free to execute this alternative without trying first the other alternatives.

Alternatives can be robust to different errors and might use quite different amounts of resources (CPU, memory, etc). They should nevertheless be marked as or alternatives as long as they provide the same functionality. The RM’s task is to select the alternative that fits best the current resource availability. In this sense, the RM can avoid errors from happening by executing alternatives whose resource requirements are more likely to be satisfied in the current environment. Some alternatives might not be equivalent from a functional point of view. For example, in Figure 8, function $g()$ has two ways to calculate its return value: (1) using a precise equation, or (2) using some approximation. Normally, we would like to use the precise way unless there is an error like a failed bound check.
In this case, the programmer of function \( g() \) knows that the alternative is less useful from a functional point of view. From a non-functional point of view, the approximation might be more efficient and less error prone and hence, would be naturally be preferred by the RM. The RM therefore needs to be able to distinguish between more and less desirable alternatives from a functionality point of view. The programmer can specify such less desirable alternatives using the keyword `or else`. In Figure 8, a RM would know that the second alternative for function \( g() \) is less desirable than the second alternative of function \( f() \). In other words, the RM can select alternatives that are not known from a local point of view, thus providing a more composable recovery.

Keywords `or` and `or else` introduce a partial order over alternative blocks. The goal of the RM is to execute alternatives that provide the highest functionality with respect to the given partial order. When the execution reaches an atomic action with `or alternative blocks, the RM is free to select which of these blocks is to be executed. Amongst these alternatives, it should execute the alternative that is expected to need the least resources. For example, the RM could, based on observations, expect that the first alternative is likely to fail and hence, might effectively need more resources than other alternative blocks. Therefore, it might always execute one of the other alternatives first.

Note that the `or else` syntax was taken from [19]. However, we do not always want to assume that alternatives are necessarily totally ordered by their utility or that they are tried in the same order as in the source code. Hence, we added `or` alternatives (also see Section 5.1).
3.2 Explicit Error Handling

Up to now, we have assumed that the recovery manager (RM) has complete control over the recovery strategy and that programmers only provide hints if they want to. Although we believe that this works in many instances, we also need to support cases in which the recovery strategy is application-specific. However, we would not like to use exceptions, error return codes, or similar mechanisms to report the errors because these suffer from the problems explained in the previous sections.

Note that we do not consider custom compensation actions (e.g., for external actions) and code that keeps track of the execution of atomic blocks (e.g., logging or statistics) to be cases of explicit error handling. The former should be a feature of or linked to the TM implementation because undo paths and error propagation paths are different. The latter should be a customizable feature of the RM runtime library because this would allow for more advanced and standardized error logging across different applications. For example, the program could provide the RM with descriptions of the functionality of atomic actions during execution, and the RM could then provide different kinds of error reporting depending on whether the application has a GUI or not.

3.2.1 Difficulties

Improving explicit error handling is difficult due to several issues, which we explain in the following paragraphs.

First, are errors part of the functional interface of an object? This of course depends on the semantics of the function but the consequences for callers also have to be considered. For example, a function that parses an FTP response string for an error code should return the code. However, a function that executes an FTP command should fail if the server reports an error because simple callers should not be required to always explicitly check return codes. Nevertheless, advanced callers should be able to retrieve the error code because they could ask the user for error-specific help (e.g., to try again in a few hours). To summarize, the mechanism for explicit error handling should be compatible with implicit error handling in the sense that specific errors should be converted into unspecified errors if the caller does not want to handle them.

Second, errors are reported in a different context than in which they are handled in. Describing an error for an error handler is difficult because the description consists of information that possibly depends on previous actions, whereas the error is to be handled after these actions have been rolled back. Thus, the programmer has to create independent information, which is difficult and can hinder composability.
Third, how do we describe errors? Error descriptions could range from simple error codes to large data structures describing every detail. Likewise, callers could prefer either detailed or coarse descriptions.

As an example, let us consider exception handling. A programmer has to select a proper type for the exception and create an exception object (e.g., Java’s `IllegalStateException`). This might not be as easy as it seems because callees and callers must agree on the meaning of exception classes: the callee must throw an exception that is meaningful for the caller and closely matches the error, and the caller must make sure that it catches all relevant exceptions. Thus, the more precise the exception-throwing callee is, the more tedious it gets for the caller to handle all cases. Hierarchies of exception classes help decrease this problem.

Besides selecting a proper exception type, the callee also has to include in the exception object any information that might be of use for the caller. However, the callee should not have to consider what is required for the caller for recovery besides what is part of the interface, because relying on this information would prevent composability.

One advantage of our approach is that most of the application-independent, hard to describe errors (e.g., bit flips) can be handled using implicit error handling (which can coexist with explicit handling). Callees should only include errors specific and relevant to the callee’s application logic in their specification. If an error has no representation at the caller’s side (either because the callee did not specify it or the caller does not want to handle it), there is also no need to describe the error because the RM will handle this error (i.e., implicit error handling).

### 3.2.2 Error Contexts

Our approach for explicit error handling is based on defining an error context which is a set of key–value pairs. A callee can set key–value pairs which can be both used by implicit and explicit error handling. Using key–value pairs is quite different from throwing exception objects or the mechanisms proposed in [18, 4].

Figure 9 shows an example. To handle errors, atomic actions can be followed by an atomic block marked `handleerror` that contains the error handling code. The error context is a set of key–value pairs that are accessible to the error handling code.

A custom error can be signaled by modifying the error context in a `fail` block, i.e., by setting one or more key value pairs. If this happens, then the atomic action will be rolled back but the error context will not. If a custom error is signaled, the RM decides if and which `handleerror` atomic block will be executed. The `handleerror` block can check which error has occurred and take appropriate actions. Note that `fail` blocks and `handleerror` blocks are atomic blocks. Error handlers can additionally signal custom errors to their callers. If signaling fails,
atomic { fail ; } // implicit error handling
2 or {
3 fail {
4 RM.set("errorCode", 23);
5 } }
6 or {
7 String str ("filename");
8 fail {
9 RM.set("desc", str ); // copies str
10 } }
11 handleerror {
12 int err ;
13 String desc;
14 if (RM.get("errorCode", err )) { print (err ); }
15 else if (RM.get("desc", desc )) { print (desc ); }
16 }

Figure 9: Example for explicit error handling.

this is equivalent to an unspecified error (see the first alternative in Figure 9).

An important distinction from other approaches is that the RM can choose to
override custom errors and thus switch to implicit error handling if this seems
beneficial. This is important because it allows to retry the atomic actions first
before executing error handling code, as well as to try other alternatives on the
caller side first. This ensures composability. It is also consistent with our basic
assumption that anything can fail eventually (e.g., due to a bit flip).

3.3 Repair Actions

Up to now, we have focused on recovery of a single atomic action. However, this
only covers faults that are in the action itself or faults that the action can deal with.

If a fault that leads to an error in the current atomic action is outside the scope
of the outermost atomic action, we cannot prevent its effect (and thus cannot fix
the fault) because we cannot roll back further than to the start of the outermost
atomic action. Similarly, the fault might be in some other component that just
shares resources or other objects with the current action.

We thus need to be able to repair the environment of the current action to be
able to let it proceed successfully. This requires the cooperation of other parts
of the application or of its environment (e.g., services and resources). The action
alone cannot know how to repair the environment because of information hiding,
and it should not have to know in order to allow composability.

In contrast, the recovery manager (RM) has all the information, so it only
needs objects to specify how they can cooperate, but not when. We call such
offers for cooperation repair actions. They modify the state or behavior of the
components in a way that hopefully decreases the chance that other actions will
encounter an error. The RM is free to execute repair actions whenever it assumes
that these might help.
As an example, consider an object that manages a cache. This cache might be in some other component, or even in another application if this is also controlled by the same RM. Conceptually, the cache object can be seen as a server running in its own thread and waiting for incoming requests. The cache uses memory just to improve performance. Now consider that the current atomic action must allocate some memory to perform its task but there is not enough memory left. As the cache can safely release its memory, it should cooperate by deleting its elements. The RM just needs to send a request for cooperation to the cache when the out-of-memory error happens in another action. For the cache, this would be just another request coming in. Perhaps counter-intuitively, implementing the cache as a concurrent object actually allows for better recovery because this gives the RM more recovery actions that it can try. In turn, atomic blocks allow programmers to implement concurrent objects safely and easily. Note that a repair action that removes all elements from the cache could also repair other faults, for example, a corrupted cache data structure. Micro-reboots [8] could also be triggered by repair actions.

If repair actions encounter an error, they simply fail. The RM is notified about failure and can adapt its strategy accordingly. Repair actions need an execution context (e.g., input data). We distinguish between two types of context in which they can execute: (1) in a new transactional context, and (2) embedded in some transactional context or other execution context. The first option is equivalent to an outermost atomic action with predetermined input data and concurrent execution, whereas the second context is either a nested action that sees all the data of the enclosing atomic block or an outermost atomic action, and gets triggered by ordinary control flow in some thread.

We call the first type independent repair actions. We constrain the input of these actions to a reference to a single object, which binds a repair action to a certain object and this object’s lifetime. Repair actions can limit their activity by signaling an error when they are not applicable at the moment of the execution. Figure 10 shows an example: the cache’s clearCache() method has an annotation that indicates it is an independent repair action.

Embedded repair actions are always executed in some execution context. They can only be executed whenever ordinary application control flow reaches them. Figure 11 shows two possible notations for embedded repair actions. Expressing them as alternatives is the first possibility and conceptually similar. However, the
second, more explicit notation might be more useful because it is easier to spot for developers. Furthermore, it explicitly informs the RM that this action is a repair action, so executing it might help globally even if the current action did not encounter any errors.

Note that embedded repair actions have a restricted utility because they do not take effect globally until their enclosing action commits, so an enclosing action can only use embedded repair actions to fix transaction-local problems. For example, it does not help to clear the cache directly in the action that fails due to an out-of-memory error because the garbage collector will not reclaim memory until the element removals in the cache are committed.

### 3.4 Integrating Traditional Exception Handling

In this section, we elaborate on how to integrate exceptions, currently the primary mechanism for error handling, with the new language mechanisms that we proposed. Exceptions are used to signal unexpected errors (e.g., in Java, exceptions that are subclasses of `RuntimeException` or `Error`) as well as to report return codes (or error codes) to the caller (e.g., `IOException`).

The first way to integrate exceptions with atomic blocks is to treat exceptions in the same way as errors signaled with `fail`. That is, they have abort semantics as in [18].

However, this prevents exceptions being used as error codes or return codes. If an atomic block fails, it gets rolled back and the exception cannot use the data seen in the atomic block to construct the exception. In Java, exceptions are often used as out-of-band return codes because a lot of basic classes (e.g., `Integer`) are immutable, there are no pointers, and methods can return only a single value. Transparently treating exceptions as failures can also break legacy code. Exceptions do not necessarily represent errors. They can also just be used to easily pass values or as another control flow construct [6]. Also, legacy code might rely on the fact that changes are not rolled back.

Because we do want to safely integrate legacy code, we propose to keep exceptions as a legacy mechanism that is not aware of transactions. Exceptions do not

```java
atomic {
  x = 10;
  // An alternative blocks construct similar to a repair action:
  atomic { } or else { performRepair(x); }
  // Explicit notation:
  repair { performRepair(x); }
  // Will roll back repair actions:
  fail;
}
```

Figure 11: Embedded repair actions.
let atomic blocks fail, and they can be used for return codes or error codes. Utilizing advanced recovery strategies and the RM when certain special exceptions are thrown (e.g., `IllegalStateException` or `OutOfMemoryError`) is still possible using this approach.

Figure 12 shows how exceptions can be caught in a safer manner. The user is notified when an exception is caught that can be handled, otherwise an error is signaled. Note that this only works because exceptions do not signal an error but can be handled.

4 Recovery Manager

In this section, we sketch the design and semantics of recovery managers (RMs). We discuss the guarantees that they provide, describe a basic RM, explain how RMs can get hints from the program about errors, and give examples for advanced recovery strategies.

RMs require an underlying implementation of atomic blocks. We assume that this implementation is transactional memory (TM). They must be able to start and abort atomic blocks (i.e., transactions) and must be notified by the TM when a transaction is aborted for concurrency control reasons.

For each atomic action, an RM must only guarantee that it finds a valid execution path through the tree of atomic blocks. We only require relaxed guarantees to allow as many recovery strategies as possible. In particular, there are no strict requirements regarding the selection strategy and execution order for alternatives. Alternatives can even be completely ignored by RMs.

4.1 A Basic Recovery Manager

A very simple strategy is to assume that atomic actions are completely independent of each other: they do not cooperate during error recovery and they do not remember the results of past recovery attempts. Explicit error handling is enabled.

```java
atomic {
  try {
  }
  catch (IOException ex) {
    notifyUser();
  }
  catch (Throwable ex) {
    fail;
  }
}
```
With such a strategy, the RM performs for each outermost atomic action a depth-first search on a tree that contains all possible execution paths through the nested atomic blocks. For each atomic block, the RM simply tries all atomic blocks in the order they have in the source code (but executes blocks with degraded functionality last). If a block fails, then the next block is tried. If execution of a block completes without error, then the atomic action succeeds. If all blocks fail including explicit error handling, then the whole atomic action fails.

This strategy is similar to classical error handling with try/catch blocks. However, the RM could also first perform an additional depth-first search but with explicit error handling disabled so as to increase the chance of finding a successful path, something that cannot be conveniently achieved using exception handling.

4.2 Symptoms

When using implicit error handling, programmers are not required to specify the type of error that occurs or its cause because that is often difficult to do correctly. Additional information can obviously help RMs to select good recovery strategies, so allowing a program to communicate this can be useful.

The problem is that an object has only a local view and may not see (and should not have to inquire) the global perspective of causal relationships between faults and errors. Conversely, the RM can reason about the big picture but does not know about the semantics of the objects and thus cannot get accurate observations at the object level.

We therefore propose that objects can optionally inform the RM about symptoms that they observe (e.g., “my calling context did not satisfy my precondition”, or the value error in Figure 6). Atomic actions can also express that an alternative might be robust to a certain symptom. Nevertheless, this information is only a hint to the RM, which is free to use or ignore it.

We have not yet investigated in detail which symptom categories would be necessary and how the RM’s interface for symptoms should look like. From the perspective of an object, examples for symptom statements could be “I don’t support this operation yet”, “Something is wrong with object O”, or “This alternative has little memory overhead”.

4.3 Advanced Recovery Strategies

We do not evaluate advanced recovery strategies in this paper but we briefly sketch some examples for strategies and how they can easily be implemented as RMs.

Graceful degradation can be implemented by or else alternative blocks or by alternatives that communicate appropriate symptoms. If the program specifies
the locally expected utility of atomic actions and the level of degradation of each alternative, an RM has enough information to find a good execution.\footnote{Note that it can also try to determine the utility, for example by measuring CPU usage.}

RMs can employ machine learning and use statistical data to build a recovery model of the application at runtime. For example, they could track which alternatives are likely to fail. Atomic actions can be tried and rolled back, which eases the learning process.

An RM can (in coordination with the scheduler for atomic blocks) execute atomic actions in complete isolation or with randomized schedules, thereby increasing the chance that transient faults do not activate errors.

Information from the environment (e.g., performance counters) or even “recovery strategy plugins” can be used by RMs because of the separation of the recovery concern from the core application code. This is important because some strategies are only required for certain environments (e.g., with high soft-error rates) or are only useful in certain environments (e.g., real-time tasks should not take too long to recover).

Finally, RMs should also provide error logging capabilities and help in creating informative error messages for users.

5 Discussion

To discuss our approach, we compare our approach with other proposals and evaluate it using the results from a recent field study on exception handling.

5.1 Comparison to Related Approaches

In this section, we compare our approach to related recent approaches. Background related work is discussed in Section 1.1.

In [7], Cabral and Marques argue that the environment of an application should provide means for automatic recovery from errors instead of requiring programmers to provide custom error recovery code. They propose to combine recovery blocks [5] with a “mini-transactional system”. However, they do not provide details about the envisioned language integration and the required runtime mechanisms. Their proposal also does not consider coordination and cooperation as provided by the recovery manager (RM) and repair actions in our approach.

Shinnar et al. describe in [4] how traditional exception handling can be enhanced by adding automatic rollback for specially marked \texttt{try} blocks. If an exception is caught in such a block, memory updates performed in the block and in the exception handler will be rolled back. If an exception handler throws a
The main difference to our approach is that they build solely upon traditional exception handling and explicit, programmer-controlled recovery. The former introduces the typical problems of exceptions and how programmers use them in practice [6]. The latter decreases composability significantly as we have argued in previous sections; for example, it requires object-local exception handlers to follow and interact with global recovery strategies, which is difficult for programmers and should be avoided where possible. We think that the concept of implicit error handling, repair actions, and having a RM that coordinates recovery globally are necessary ingredients for providing error recovery that is automatic, truly composable, and easier to use by programmers.

The retry, do, andorElse constructs [19] proposed by Harris et al. can be used to implement the concepts of blocking and choice in a composable manner. However, they are not well-suited to implement error recovery due to an important conceptual reason as well as details in their specification.

First, blocking for an event (or a set of events) to happen is conceptually different from signaling an error. The former expresses that it is correct for the program to wait until the event occurs, no matter how long that might take. When signaling an error, the program basically states that it cannot perform its function anymore. Although it might be useful to retry the operation a couple of times, the program must be allowed to terminate in case of a fatal error. This is not compatible with the concept of blocking, so both retry and fail should be provided to programmers.

Second, the details of the semantics of orElse are targeted toward blocking. If the first alternative of two retries, the second alternative has to be tried, and if the second retries as well, the TM should wait for changes in both alternatives’ input data. This fixed, total order of execution is useful to easily build nonblocking variants of blocking code as [19] shows. However, this does not allow the programmer to specify truly equivalent alternatives; even if the first alternative is likely to fail, a RM would have to always execute the first alternative before executing the second. This is why our approach has or blocks and allows RMs to decide which alternative is to be executed.

5.2 Our Approach vs. Exception Handling

Cabral and Marques performed a field study [6] on exception handling in libraries, server and stand-alone applications written in Java and .NET. They state that normally, programmers use exception handling to inform the user, abort the application or force the application to continue, but not to actually recover from errors. Our approach is instead focused on recovery rather than just detecting and report-
Table 1: A summary of error handler categories from [6] and how these errors can be taken care of in our approach.

### Category Description and typical usage Counterparts in our approach Automatic/generic handling possible? Composability improved by our approach?

<table>
<thead>
<tr>
<th>Category</th>
<th>Description and typical usage</th>
<th>Counterparts in our approach</th>
<th>Automatic/generic handling possible?</th>
<th>Composability improved by our approach?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty</td>
<td>No error handling. Often used to avoid termination on non-fatal exceptions (especially in .NET) and to work around error handling requirements (in case of checked exceptions). Another typical use is implementing alternatives.</td>
<td>Alternatives with same or different error handling. Custom error handling can also be defined.</td>
<td>No because assertions are checked. Often used in .NET.</td>
<td>No because assertions depend on the application logic.</td>
</tr>
<tr>
<td>Try</td>
<td>An exception or another error is thrown.</td>
<td>Try blocks provide built-in recovery strategies. RM can combine user text for “Close”.</td>
<td>Possibly (implicit error handling).</td>
<td>Yes. RM can combine user text for “Close”.</td>
</tr>
<tr>
<td>Continue</td>
<td>Protected code is in the body of a loop. The handler starts the next iteration of the loop.</td>
<td>Depends on the exact details of the application code. The RM can terminate applications and log errors.</td>
<td>No because this is part of the application logic.</td>
<td>Yes. The RM also has a lot easier access to more runtime information (e.g., number of retries).</td>
</tr>
<tr>
<td>Return</td>
<td>Return from method or terminate the application. Likely to be paired with logging in server and stand-alone apps.</td>
<td>If method return is used to terminate non-fatal errors, get a less verbose block.</td>
<td>No because this is part of the application logic.</td>
<td>Yes. The RM can keep history of errors and execution, and thus does not lose information.</td>
</tr>
<tr>
<td>Rollback, Close</td>
<td>Rolling back and rolling closing database connections, releasing locks.</td>
<td>Atomic blocks provide built-in rollback and cleanup support. TM can be used instead of programmer-controlled rollback.</td>
<td>No because this is part of the application logic.</td>
<td>Yes in case of retry.</td>
</tr>
<tr>
<td>Audit</td>
<td>Assertions are checked. Often used in .NET.</td>
<td>No because assertions depend on the application logic.</td>
<td>Possibly because RM controls the execution and could try to avoid transient faults.</td>
<td></td>
</tr>
</tbody>
</table>
proach can provide a generic solution or automatic error handling, which means that programmers would not have to write the handler but could rely on the recovery manager (RM) to perform this task. The fifth column shows whether our approach increases composability for a particular kind of error handling. Please note that whenever we can provide generic support for a certain handler type, this also increases composability in practice (e.g., standardized logging).

A percentage of 60% to 75% of all handlers belong to the “Empty”, “Log”, or “Alternative” categories. A significant part of handlers in servers is in the “Others” category (not shown in the table), but the study states that the handling mostly consists of cleanup actions (similar to “Close”).

One major reason why we cannot support generic support for some error handling objectives is that we cannot automatically detect or create alternatives. This is application-logic–specific and has to be specified by the programmer. However, the RM can try to automatically find a path through possible alternatives, which, assuming a proper alternative specification by the programmer, results in cleaner code in our opinion.

One shortcoming of our approach is that it cannot yet provide automatic cleanup for actions that have already been committed. According to the study, a majority of finally handlers is in the “Close” category. One could gain automatic cleanup by enlarging atomic actions so that they cover sufficiently large portions of the program. However, this can have a negative effect on performance and can make the execution of some external actions more difficult (see [34] for a detailed explanation). This problem is not specific to our approach though but rather a general trade-off when using transactions. We think that techniques along the lines of [34] together with integration into the RM could overcome this limitation.

The study does not directly reveal how often programmers intend different handlers to be executed for different errors occurring in the same action. This can determine whether programmers would have to use explicit error handling or could rely on implicit error handling. 50% to 80% of the exception handlers in .NET code catch all errors and thus probably allow implicit error handling to be used. In Java, programmers also prefer the most generic exception classes. Note that even when a specific exception is caught, the handler could still be replaced by implicit error handling if the handling performs generic tasks (e.g., all errors can be logged and only the log output can be filtered later on depending on the actual type of error). In contrast, the types of errors that will be handled (and thus signaled) are much more specific and varied. This is positive for our approach because it indicates that programmers can provide the RM with details about the error, which helps the RM making decisions. There are two other results that potentially indicate that implicit error handling can be used often, although they are not conclusive: (1) on average, only one handler is associated with a try block and (2) only 30% to 55% of the try blocks have associated catch blocks.
To summarize, the results of the study suggest that our approach increases composable in case of most errors. Our approach also enables generic/automatic support for a large part of the error handlers, which further increases composable, and implicit error handling seems to be sufficient often.

6 Conclusion

Current wide-spread approaches to recovery such as exception handling do not give programmers a robust and easy way to compose objects or components because when using exceptions or explicit error handling, programmers have to implement the recovery strategy (i.e., the big picture) in the application’s components. Furthermore, strategies in different components must match to be able to recover.

However, an object should only specify in which ways it is able to cooperate or recover locally. It should not have to think about the big picture nor assume a certain cooperation and recovery scheme.

In this paper, we presented an approach that combines atomicity and separation of the recovery strategy from core application logic to make error recovery composable and flexible with respect to recovery strategies. Because of the better separation, recovery strategies can be plugins that are specific to certain environments, can potentially perform cross-application recovery, and can use environment-specific information without support from the application itself.

We introduced recovery managers as coordinators of recovery strategies. We also presented a programming language interface for recovery that permits the separation of recovery strategies, implicit and explicit error handling, and can integrate previous error handling and exception handling mechanisms. Furthermore, we proposed repair actions, a compact interface for specifying recovery cooperation between objects.

Neither transactions nor the various existing recovery strategies are new concepts but still they are often not used in a lot of applications, or are only used if this cannot be avoided. This is partially due to the lack of composable when combining these strategies with current means for exception and error handling. We believe that the approach presented in this paper will help in making the good recovery strategies that exist today applicable to more programs and will make it easier for developers to reason about and implement error recovery.

We also hope that the approach will spawn future research in this direction, both towards recovery strategies using transactional memory and their impact, and towards further refinement of the programming language interfaces.
References


Abstract

In the context of model-driven software development, the concepts of visual modelling and model transformation are of increasing importance. Although there is a wide range of tool support for designing and executing visual models and model transformations during system design, the analysis of model behaviour and of model transformation properties is not yet supported in a satisfactory way. In this article, we consider typed attributed graph transformation as formal framework for model transformations to deal with this problem. We give a conceptual overview on research activities and a survey of current research papers on the formalization of visual modelling and analysis of model transformations based on typed attributed graph transformation.
1 Introduction

The central question of software engineering has always been how to produce software of quality. For about 40 years, programming has been the key task, meaning expensive work of highly skilled programmers. Ambitious programming projects resulted in failure, went over their budgets or proved to be unstable over time. It is common knowledge today that the need for a sound software production has to be faced from a different perspective: model-driven development (MDD). The objective of MDD is generating code from a higher-level system model. This means that, for software developers the abstraction level is now raised. No longer do they need to worry about technical details and features of programming languages but can concentrate on more creative parts of software engineering: analysis, design and validation. Different kinds of model transformations are proposed that cover the steps of a sound software production process, including business modelling, requirements engineering, conceptual modelling and model-based code generation. The issue is “the model is the code” rather than “the code being the model” [59]. Hence, modelling as key skill and model transformations as main technology of MDD become major issues in software engineering.

MDD promotes the extensive and systematic use of models from a very early phase of the design cycle. System requirements and design are captured by high-level, visual engineering models (using popular and standardized modeling languages like UML, SysML, AADL, or BPMN). Formal analysis of design models can be carried out by generating appropriate mathematical models by automated model transformations. Problems found out by automated analysis can be fixed by model refinement prior to implementation. Finally, the source code of the target system can be derived by automatic code generation.

Unfortunately, MDD techniques are not yet mature enough to completely close the “semantic gap” between model notation and programming language. For instance, many modeling languages are not formal enough to create anything more than code skeletons. Behavioural modelling languages, in particular, should be precise enough to allow for a translation ensuring behaviour equivalence.

On the one hand, there is a wide range of tool support for designing and executing model transformations during system design (with popular languages like QVT [58], ATL [41, 40] or graph transformation [20, 21]). However, most model transformations are still written manually in industrial practice as a regular piece of software in a rather ad-hoc way. Thus, even automated model transformations can be erroneous and might invalidate the results of a thorough mathematical analysis, i.e. when problems are reported during mathematical analysis, it is impossi-

1The automated transformation process is also referred to as “correct-by-construction” as opposed to “construct-by-correction” [67].
ble to distinguish whether they are due to erroneous system design or to a flaw in the model transformation.

As formal framework to deal with some of these problems, we choose typed, attributed graph transformation, a formally defined calculus based on graphs and graph transformation rules [20, 19]. For ages, rules have proven to be extremely useful for describing computations by local transformations. Areas like language definition, logic, functional programming, algebraic specification, term rewriting and expert systems have rules as key concepts. Graph transformation, also known as graph rewriting or graph reduction, combines the potential and advantages of both graphs and rules into a single computational paradigm. Nearly 40 years ago, Rosenfeld et al. [65, 60] in the USA and Schneider, Ehrig and Pfender [26] in Europe introduced graph transformation for generation, manipulation, recognition and evaluation of graphs. Since then, graph transformation has been studied in a variety of approaches, motivated by application domains such as pattern recognition, semantics of programming and modelling languages, specification of distributed systems etc. [21, 25, 5].

A detailed presentation of different graph transformation approaches, is given in volume 1 of the *Handbook of Graph Grammars and Computing by Graph Transformation* [66]. The algebraic approach is based on pushout constructions, where pushouts are used to model the gluing of graphs. In fact, there are two main variants of the algebraic approach, the double and the single pushout approach. The double pushout (DPO) approach [20], is the formal basis for visual modelling of behavioural models and model transformations considered in this article. The DPO approach is based on category theory: a graph transformation rule is a pair of morphisms in the category of graphs with total graph morphisms as arrows: \( r = (L \leftarrow K \rightarrow R) \) (or \( L \supseteq K \subseteq R \)) where \( K \rightarrow L \) is injective. Graph \( K \) is called *gluing graph*. Another graph morphism \( m: L \rightarrow G \) models an occurrence of \( L \) in \( G \) and is called a *match*. Practical understanding of this is that \( L \) is a subgraph that is matched to \( G \), and after a match is found, the rule can be applied. A direct transformation or application of rule \( r \) to graph \( G \) is defined by two pushout diagrams (see the diagram to the right). Applying the rule, \( m(L) \) is replaced with \( m'(R) \) in graph \( G \), leading to the transformed graph \( H \). A graph transformation, or, more precisely, a graph transformation sequence, consists of zero or more direct transformations, written \( G_0 \Rightarrow G_n \). A set of graph rules is called graph transformation system. A type graph defines a set of types which can be used to assign a type to the nodes and edges of a graph. The typing itself is done by a graph morphism from the graph to the type graph. A typed graph transformation system \( GTS = (TG, P) \) consists of a type graph \( TG \) and a set \( P \) of typed graph rules.
A (typed) graph grammar $GG = (GTS, S)$ consists of a (typed) graph transformation system $GTS$ and a (typed) start graph $S$. The (typed) graph language $L$ of $GG$ is defined by $L = \{ G \mid \exists$ (typed) graph transformation $S \Rightarrow G \}$. The key idea of attributed graph transformation is to model graphs with node and edge attributes, i.e. an attributed graph is a pair $AG = (G, A)$ of a graph $G$ and a data type algebra $A$. Typed attributed graph transformation [19], combining process and data modelling proved to be well-suited to define and analyse visual models and model transformations [20, 53].

A variety of tools for graph transformation exist [75] to be used as transformation engine and for analysis purposes, to reason about issues such as conflicts and dependencies of actions as well as consistency of object structures. In this article, we discuss the formalization of visual models and model transformation by typed attributed graph transformation and introduce formal techniques and tools to analyse model properties and model transformation correctness.

The article is structured as follows: section 2 introduces different approaches for formalizing visual modelling and model transformations by graph transformation. In section 3, the analysis of model transformations is considered. We conclude in section 4 with an outlook on future lines of research.

2 Visual Modelling and Model Transformations

In this section, we introduce different approaches for formalizing visual modelling and model transformations by typed attributed graph transformation. After an overview of modelling approaches used in MDD, we build a bridge from meta-modelling to the graph transformation and discuss a formal view concept for structuring visual models and meta-models. Concerning the modelling of model transformations, we rely on graph transformation rules and suitable rule application control structures as formal basis and compare existing approaches.

2.1 Model-Driven Software Development

In the context of MDD, domain-specific modeling languages (DSMLs) are proposed defining the application structure, behaviour, and requirements within particular domains. DSMLs are described by meta-models, where relationships among concepts in a domain are given by class diagrams and language properties by constraints using OCL [56] associated with these domain concepts. A meta-model describes the various kinds of model elements of a DSML, and the way they are arranged, related, and constrained. As models, meta-models are also composed of elements. A model conforms to a meta-model if each model element has its meta-element defined within the meta-model, and the given constraints are satisfied.
The Bulletin of the EATCS

For structuring meta-models, meta-meta-models are used. In the same way models are defined in conformance with their meta-model, meta-models are defined by means of the meta-meta-model language. An overview of different levels of meta-modelling is given in Figure 1, where the model is a Petri net, the meta-model a class diagram defining the structural knowledge of Petri net concepts (Place, tokens, Transition and arc), and the meta-meta-model shows the key model elements for modelling class diagrams (the meta-model language).

2.2 From Meta-Modelling to Graph Grammars

While meta-modelling provides a declarative approach to DSML definition, grammars are more constructive, i.e. closer to the implementation. Due to its appealing visual form, graph grammars can directly be used as high-level visual specification mechanism for DSMLs [4]. Defining the abstract syntax of visual models as graphs, a graph grammar defines directly the DSML grammar by graph transformation rules. The induced graph language determines the corresponding DSML. Visual language parsers can be immediately deduced from such a graph grammar. Furthermore, abstract syntax graphs are also the starting point for visual modelling of model behaviour [31, 11, 77, 39] and model transformations [53, 52, 71, 68].

Meta-modelling is closely related to graph typing where a type graph takes the role of the meta-model, and an instance graph, typed over the type graph, corresponds to a model conforming to a meta-model. In order to better map meta-modelling concepts to typed, attributed graphs, the graph transformation theory has been enhanced in [4] with node type inheritance facilities, and it has been shown how typed graph transformation with inheritance can be flattened to simple typed graph transformation. Meta-modelling and graph transformation can be integrated by identifying symbol classes with node types and associations with
edge types. Table 1 shows a comparison of main meta-modelling notions to their counterparts in the terminology of typed graphs.

<table>
<thead>
<tr>
<th>Meta-Modelling</th>
<th>Modelling by Typed Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meta-model</td>
<td>Type graph $TG$ with attribution, inheritance, multiplicities.</td>
</tr>
<tr>
<td>Instance Model</td>
<td>Attributed graph $G$, typed over type graph $TG$</td>
</tr>
<tr>
<td>Class</td>
<td>Node in type graph $TG$</td>
</tr>
<tr>
<td>Object</td>
<td>Node in $TG$-typed graph $G$</td>
</tr>
<tr>
<td>Association</td>
<td>Edge in type graph $TG$ (with possible multiplicities)</td>
</tr>
<tr>
<td>Reference</td>
<td>Edge in typed graph $G$ that must not violate the multiplicity constraints in $TG$.</td>
</tr>
</tbody>
</table>

Table 1: Mapping meta-modelling notions to graph terminology

The notion of being a model conforming to a meta-model can be adequately formalized for typed graphs by the existence of a typing morphism from an instance graph to the type graph. In Figure 1, the Petri net (the model) is typed over the type graph (the meta-model), since we have a typing morphism from the Petri net to the type graph, as indicated by the arrows which map all Place nodes to type Place, and the Transition node to type Transition. Tokens are Place attributes. Their values (the token numbers within the Place nodes) are mapped to the data type integer.

OCL constraints can be translated to graph constraints or to graph transformation rules (syntax rules), as has been shown in [79, 9]. Thus, declarative as well as constructive elements may be used for DSML definition based on typed graph transformation. An example for a rule from the syntax grammar for Petri nets is shown in Figure 2, where an arc is inserted between a place and a transition node. Note that the graphs in this rule conform to the meta-model (are typed over the type graph) shown in the center of Figure 1.

![Figure 2: Syntax rule adding a Petri net arc](image)

Multiple modeling notations and interpretations (views) are helpful to represent different aspects of concern and different abstraction levels. In the context of MDD, different views of a system can be expressed by having different meta-models which may be related, nested or partly overlapping. For example,
connection from class diagrams to behavioural aspects is given by a method signature section that can be defined for a class. Moreover, UML state diagrams relate data aspects and behavioural aspects of objects.

UML's many loosely connected kinds of diagrams make it so difficult to define a rigorous semantics for full UML. With respect to UML, formal approaches based on graph transformation exist aiming to relate (at least a part of) the model elements of various views by defining the combination of messages, objects, states, events, data values and so on, in one integrated model, whose behaviour then might be simulated: in [51, 34, 44, 29, 77], different UML diagram types are given a well-founded semantics in terms of graph transformation systems, and in [45, 81], an integrated formal semantics of a UML model consisting of various different UML diagrams has been defined as graph transformation system.

Different abstraction levels of models can be expressed using meta-modelling by different hierarchy levels of meta-models, each conforming to the meta-model of the next-higher level (similar to the meta-model hierarchy shown in Figure 1).

Using graph transformation, we can precisely define a view as graph morphism embedding a view type graph into a larger one, and we define a hierarchies of abstraction levels by typing graph morphisms from the lower level's type graphs to the more abstract ones. Extending the formal framework of typed graphs and graph morphisms (the category \texttt{Graphs}_TG), the structuring concept of \textit{views} for visual models is formalized in [18], where typed attributed graph morphisms now take into account changes of the underlying type graphs. On this basis, formal conditions for the integration and decomposition of consistent views are given.

### 2.3 Model Transformation by Graph Transformation

In MDD, model transformations occur for various purposes. The MDA core idea [35] is mapping platform independent model level concepts (PIM) to platform specific ones (PSM) like code, XMI documents, or database schemes. Beyond this kind of transformations, another dimension of complexity appears when we want to make semantic driven transformations. Examples for this kind of transformations are: design pattern applications [73], model refactorings of UML models [74], or specialization of models of product lines.

Until now, model transformations have in most cases been developed within modeling tools using tool-specific proprietary languages. This tool adherence jeopardizes the reusability of domain models. A second approach corresponds mainly to the tree-transformation systems such as XSLT [80]. An XSLT script declaratively specifies how to explore the input tree and how to generate fragments of the output tree. XSLT is heavily used in some domains, but experiences showed shortcomings with respect to modularity, efficiency, reusability and maintainability. Hence, complex model transformations should rather be specified us-
In the classical approach to model transformation by graph transformation [13], an integrated meta-model $TG_i$ consists of the meta-models for the source and target DSML, and, additionally, reference nodes with arcs mapping source nodes to target nodes. Based on $TG_i$, graph transformation theory allows us to express model transformations directly by graph transformation rules $L \leftarrow K \rightarrow R$ with $L$ representing source model elements, and $R$ representing the corresponding target model elements, where $L, K$ and $R$ are typed over $TG_i$. The model transformation starts with graph $G_S$ typed over $TG_S$. As $TG_S$ is a subgraph of $TG_i$, $G_S$ is also typed over $TG_i$. During the model transformation process the intermediate graphs are all typed over $TG_i$. After applying non-deleting model transformation rules $r_1, ..., r_n$ the resulting graph $G_n$ is typed over $TG_i$. To delete all items in $G_n$ which are not typed over $TG_i$ we can either provide deletion rules or construct a restriction (a pullback in $\text{Graphs}_{TG_i}$), which deletes all these items in one step (see the diagram to the right).

See [17] for a full case study of a model transformation from activity diagrams to Petri nets by the classical DPO approach to graph transformation. We implemented this approach in our tool AGG [76, 7], supporting the definition of type graphs, typed attributed graph rules and constraints. Further graph transformation systems for domain-specific model transformations are VIATRA2 [3] and the Graph Rewriting and Transformation Language (GReAT) [71]. By VIATRA2, developers define graph patterns and graph transformation rules as components using a textual domain-specific programming language. The components are assembled into complex model transformations by abstract state machine rules. In GReAT, meta-models the source and target models are used to establish the vocabulary of $L$ and $R$ and to ensure that the transformation produces a well-formed target model. A comprehensive overview on existing MDD tools supporting model transformation is given in [70].

The classical approach has been extended to support the transformation of EMF models in Eclipse, thus bridging the gap between MDD tools and those for graph transformation: The Eclipse Modeling Framework (EMF) [12] has evolved to one of the standard technologies to define modeling languages. EMF is based on MOF [57] and provides a (meta-)modeling and code generation framework for Eclipse applications based on structured data models. Containment relations, i.e. aggregations, define an ownership relation between objects. Thereby, they induce a tree structure in instance models, implying some constraints that must be ensured at run-time. As semantical constraints for containment edges, the MOF specifi-
The Bulletin of the EATCS

cation states that "an object may have at most one container", and that "cyclic containment is invalid". A transformation framework for EMF models is presented in [6], where containment edges are modelled as graph edges of a special containment type. The problem is to guarantee that EMF model transformations defined by graph transformation always satisfy the EMF containment constraints. In [8], these constraints are translated to a special kind of EMF model transformation rules such that their application leads to consistent transformation results only. EMF model transformation is supported by our tool EMF Tiger [7].

Another model transformation approach based on graph transformation are triple graph grammars (TGGs) [68] which transform pairs of related models simultaneously while maintaining their consistency. TGGs generate languages of triple graphs, consisting of a source graph $G^S$ and a target graph $G^T$, together with a correspondence graph $G^C$ "between" them. A triple graph is typed by a meta-model triple which contains the source and target meta-models, and declares the types of mappings between the elements of both languages. A triple rule $tr$ consists of triple graphs $L = (S L \leftarrow CL \rightarrow TL)$ and $R = (S R \leftarrow CR \rightarrow TR)$, and an injective triple graph morphism $tr = (s, c, t) : L \rightarrow R$. A given triple graph $G = (SG \leftarrow CG \rightarrow TG)$ and a triple match $m = (sm, cm, tm) : L \rightarrow G$, a direct triple graph transformation $G \xrightarrow{tr, m} H$ from $G$ to triple graph $H$ is given by three pushouts $(SH, s', sn), (CH, c', cn)$ and $(TH, t', tn)$ in category Graph with induced morphisms $s_H : CH \rightarrow SH$ and $t_H : CH \rightarrow TH$, as shown in the diagram to the right.

Since 1994, several extensions of the original TGG definitions have been published [69, 43, 36], and various kinds of applications have been presented [75, 37, 42]. From a TGG, operational rules can be derived. In particular, we can deduce from each triple rule $tr$ a source rule $tr_S$ with empty connection and target component (analogously, a target rule can be derived with empty source component). For source-to-target (forward) model transformation, forward rules are derived which take the source graph as input and produce a corresponding target graph (analogously, backward rules can be derived):

- Source rule $tr_S$ for $L_S = (SL \leftarrow CL \rightarrow TL)$:
  \[
  L_S = (SL \leftarrow CL \rightarrow TL)
  \]
  \[
  R_S = (SR \leftarrow CR \rightarrow TR)
  \]

- Forward rule $tr_T$ for $L_T = (LR \leftarrow CL \rightarrow TL)$:
  \[
  L_T = (SR \leftarrow CR \rightarrow TR)
  \]
  \[
  R_T = (SR \leftarrow CR \rightarrow TR)
  \]

A case study applying TGGs modelling a model transformation from class diagrams to relational database models is presented in [16, 24, 23]. The relationship of TGG-based model transformations with the classical approach based on plain graph transformation has been analyzed in [22], giving a flattening construction from TGGs to typed, attributed graph transformation systems.
3 Analysis of Model Transformations by Graph Transformation

Whereas there exist several approaches for specifying model transformations in MDD in a declarative way [40, 58, 1]), there is a lack of methods for analysing declarative transformation specifications which also take into account the relations and constraints expressed by the transformation, as well as the meta-models and their well-formedness rules. In this section, we distinguish validation from verification of model transformations. Validation approaches clarify the question “is it the right transformation?” by allowing designers to test if the transformation behaves as expected. The verification of transformations aims to answer the question “is the transformation right?”, i.e. are there any errors in the transformation?

3.1 Validation of Model Transformations

Basically, validation is the ability to simulate (execute) the transformation: given a source model provided by the designer, generate the corresponding target model. For declarative approaches, this is not trivial because it is defined what is the target model corresponding to a source model, without focusing on how it is computed. Model transformations defined by graph transformation are constructive by nature and allow designers to simulate model transformation runs quite easily. Tool support for defining and simulating model transformations by controlled graph or EMF transformation is provided by our tools AGG and EMF Tiger [7, 8]. The tools also support graph constraint checking (optionally stated for single rules or for complete transformations). For rule application control, rule layers, priorities, sequences or loops can be defined. For model transformations based on TGGs, the transformation runs must satisfy the condition of source consistency in order to be valid. In [23], we present a construction for model transformation sequences which always yields consistent sequences.

Another form of validation is test case generation of source models to make simulation more systematic. EMF Tiger [7] provides suitable support for test case generation as generation rule applications can be embedded into Java code and matches can be partially defined to control the test case generation. Küster et al. [46, 28] present a white-box approach for testing model transformations by using meta-model coverage techniques, and constructing test cases driven by meta-models and constraints. Baldan et al. [2] propose a technique for generating test cases for code generators. Their approach allows the generation of test cases for testing the source code with coverage of both individual rules and rule interactions but it requires the code generator under test to be fully specified with graph transformation rules.
3.2 Verification of Model Transformations

Verification is a formal process used to evaluate whether or not a model transformation complies with certain correctness criteria, expressed by formal correctness conditions. We distinguish between syntactical and semantical correctness, as well as functional behaviour of model transformations. Syntactical correctness of target models comprises issues like meta-model-conformance (correct typing) and well-formedness (satisfying additional constraints of the target language). A model transformation behaves functionally, if we have termination (a model transformation run will always terminate for any source model) and determinism or confluence (for a given source model, a model transformation always yields a unique result). In order to be semantically correct, a model transformation should lead to target models which behave equivalently w.r.t. the corresponding source models. This is an important property of e.g. code generators for behavioural models. In the case that a model is more abstract than the code, semantical properties are defined explicitly, and it has to be shown that the properties are fulfilled by the respective pairs of source and target models.

Syntactical Correctness

Meta-model conformance in the graph transformation framework means that the target model of a model transformation is typed over the target type graph. This is guaranteed by our construction in the classical approach (see diagram on page ), where the intermediate model transformation result, which is typed over the integrated type graph, is restricted to the target type graph by a pullback construction. Analogously, the target model in the TGG approach is obtained. Concerning well-formedness, graph transformation provides means to ensure the satisfiability of graph constraints by translating them into pre-conditions for graph transformation rules [64]. In [16], conditions for TGG-based model transformations are stated under which corresponding forward and backward transformations are inverse to each other in the sense of information preservation, i.e. there is a backward transformation leading back to the same source structure as the original one. It is sufficient to show that a model transformation is source-consistent, i.e. the source component of the resulting triple graph can be constructed using source rules only. In [23], a method is described to construct source-consistent transformations on-the-fly instead of analysing consistency of completed model transformations.

Functional Behaviour

Graph transformation theory also provides powerful static analysis techniques to verify the functional behaviour of model transformations. Termination has been investigated by means of static analysis e.g. in [78, 14]. Recently, research results
have been formulated for static conflict and dependency detection, and confluence analysis for typed, attributed graph transformation with negative application conditions (NACs) [49, 48, 47]. In these works, the well-known notion of critical pairs allows for a static conflict detection. The analysis is supported by the tool AGG [7]. The formal results of conflict analysis are used in [50] to define sufficient criteria for the applicability and non-applicability of rule sequences. Completeness and correctness of TGG-based model transformations w.r.t. source consistency have been studied in [22]. In [24], the formal results were extended to TGGs with NACs (see also [69]). TGG-based model transformations with NACs were also analysed in [27] for a restricted class of triple rules with distinct kernel elements. For this restricted class, local confluence and termination can be analysed and thus, TGG-based model transformations can be checked for functional behavior.

Semantical Correctness

First static analysis techniques for behavioural equivalence of source and target models are proposed e.g. in [32, 15]. The techniques assume the operational semantics of source and target language to be given by graph transformation systems (simulation rules). The semantical equivalence is shown by applying the model transformation also to the simulation rules. A related static analysis technique verifying semantic preservation has been proposed by Rangel et al. for the borrowed context approach applied to refactoring in [61].

There are also approaches to translate the graph transformation-based model transformation into another modelling language, such as Petri nets with static analysis techniques to verify liveness and safety properties.

As an alternative to static analysis techniques, it is possible to revert to dynamic techniques like model checking for graph transformation as described e.g. in [10, 63]. These techniques have the advantage that it is possible to analyse (as e.g. in [30]) more straightforwardly any property which can be expressed by temporal logic. In many cases though the state space of rule-based models using graph transformation becomes too large or even infinite, and in this case model checking techniques have their limitations.

Karsai and Narayanan [55] establish (weak) bisimulation between source and target model of a model transformation. The approach has been extended in [54] taking into account structural correspondences in the bisimulation relation.

Automated theorem proving can support the verification of properties of rule-based models using graph transformation, e.g. by Isabelle/HOL [72, 33]. Theorem provers support also proof reasoning for models with infinite state space. Habel and Pennemann [38] address the problem to prove the validity of statements over graph-based models. Properties to be proved are given as high-level structural nested graph conditions which are as expressive as first-order logic.
The Bulletin of the EATCS

4 Conclusion

This article gives an overview of recent research results on visual modelling and analysis techniques for model transformations based on graph transformation. An important subject to be dealt with in future research will be consistent model evolution (also called meta-model/model co-evolution or model migration). Moreover, analysis techniques should be generalized to the more expressive techniques of nested constraints [38] and rules [62], as well as to controlled graph transformations. There is already some work on how to combine the advantages of both static and dynamic approaches into more powerful analysis strategies for graph transformation. Static analysis techniques may lead to property-preserving abstractions of rule-based models with infinite state space such that model checking can be applied successfully to the model abstraction.

References


THE NATURAL COMPUTING COLUMN

BY

GRZEGORZ ROZENBERG

Leiden University, Leiden Center for Natural Computing
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
rozenber@liacs.nl

QUANTUM PSEUDO-TELEPATHY SAVES THE WORLD

Michael Main
Department of Computer Science
University of Colorado
Boulder, CO 80309-0430
main@colorado.edu

Current-day theoretical physicists and computer scientists spend a good part of their lives devising thought experiments about tests that aliens might someday present to mankind. Admittedly, the scientists may actually be thinking up these outrageous tests for other reasons, such as providing a glimpse of some extraordinary properties of quantum computing, but never mind. I’d like to step through one of these tests, and to do so, we may as well assume that a motherly alien named Dhaphod has agreed to give us an advanced peek at such a test…
The Test

"When the alien testers arrive," said Dhaphod, "they will ask you to provide them with three specimens named Alice, Bob and Charlie."

Fortunately, I had three students with these names in my class, and I asked Dhaphod whether she would explain the test to my students. "Certainly," she said. She seemed anxious to please, wanting to give humankind a leg up if she could. So, I brought my three students, and Dhaphod materialized a stone tablet with chiseled bullet points:

- Before the test begins, Alice, Bob and Charlie may confer with each other to settle on the strategy that they’ll use in the test.

- At the start of the test, three aliens will take the test-takers to far flung locations. Alice will go to Rigel; Bob will go to Sirius; and Charlie will go to Thuban (the brightest star in Draco the Snake). Once they arrive at their destinations, communication between Alice, Bob and Charlie will take a long time. For example, messages traveling at the speed of light between Sirius and Thuban take 312 years, and each of the other communication pairs takes even longer.

- With everyone at their distant stars, the aliens will simultaneously give a single boulder to each human. Either all three boulders are smooth (the 3-smooth case) or exactly one boulder is smooth and the other two are jagged (the 1-smooth case).

- For the test, each human must decide whether to keep their boulder or give the boulder back to the aliens. And they must make their decisions within 42 seconds, so there is no possibility of speed-of-light communication with each other before deciding. For example, Alice cannot tell Bob and Charlie about the shape of her boulder.

- Two rules determine whether we pass or fail:

  1. In order to pass the test in the 3-smooth case, the three Earthlings must keep an odd number of boulders. This means that they must keep all three boulders or just one of the three boulders. If, instead, they keep zero or two boulders, the test is failed.

---

1 This is not as surprising as it seems. See en.wikipedia.org/wiki/Alice_and_Bob
2 According to www.shatters.net/celestia, a site where you can simulate travel to any one of thousands of nearby stars
2. For the 1-smooth case, they must keep an even number of boulders. In other words, they must keep zero or two of them.

Alice drew up a diagram (Figure 1) to help everyone understand the rules. The diagram shows four possible situations that they might face, and Dhaphod confirmed that she had it right.

"Can we take the tests many times?" Bob asked hopefully. If they could take the test over and over again, they would eventually hit a right answer just by chance.

But that was not to be. "One chance," said Dhaphod, fixing one of her eyes on each of the Earthlings. "And it’s all or nothing on that one test."

There Is No Winning Classical Strategy

Alice, Bob and Charlie sat down to work out their strategy. With the fate of Earth in their hands, they wanted to find a strategy that guaranteed passing the test, and they started with an approach that Dhaphod called classical deterministic strategies.

A classical deterministic strategy requires the Earthlings to decide ahead of time what each will do with a smooth boulder and what each will do with a jagged boulder. For example, all three might agree to keep their boulders, regardless of whether they are smooth or jagged. This strategy passes the test in the 3-smooth case (because that case requires an odd number of boulders to be kept). However, this strategy always fails the 1-smooth case (which requires keeping an even number).

It was Alice who proved that every classical deterministic strategy is imperfect. "Here’s why," she said. "For any such strategy, we can define six numbers, each of which is either zero or one." Dhaphod produced a shimmering surface and told her to write the number definitions on it with her finger. She started with the first two numbers:

- $A_{\text{smooth}}$ is zero if the strategy calls for Alice to give back any smooth boulder she is given; otherwise $A_{\text{smooth}}$ is one.
- $A_{\text{jagged}}$ is zero if the strategy calls for Alice to give back any jagged boulder she is given; otherwise $A_{\text{jagged}}$ is one.

---

3The proof is a variation on the parity game proof in Section 3.3 of "Quantum Pseudo-Telepathy" by Gilles Brassard, Anne Broadbent and Alain Tapp, Foundations of Physics 35(11), November 2005, pp. 1877-1907. We’ll see more of this paper shortly.
In the 3-Smooth Case, they each get a smooth boulder

... and they must keep an odd number of boulders.

In the 1-Smooth Case, only one gets a smooth boulder

... and they must keep an even number of boulders.

Figure 1: Alice’s Drawing of the Possible Tests
"The Bulletin of the EATCS"

"The other four numbers, \( B_{\text{SMOOTH}} \), \( B_{\text{JAGGED}} \), \( C_{\text{SMOOTH}} \) and \( C_{\text{JAGGED}} \), are defined in the same way for Bob and Charlie," she said.

"Okay," Charlie said. "I understand the six numbers. So what?"

Alice said, "So this: If a classical strategy always works, then we can write four equations..."

\[
\begin{align*}
A_{\text{SMOOTH}} + B_{\text{SMOOTH}} + C_{\text{SMOOTH}} &= \text{some odd number} \\
A_{\text{SMOOTH}} + B_{\text{JAGGED}} + C_{\text{JAGGED}} &= \text{some even number} \\
A_{\text{JAGGED}} + B_{\text{SMOOTH}} + C_{\text{JAGGED}} &= \text{some even number} \\
A_{\text{JAGGED}} + B_{\text{JAGGED}} + C_{\text{SMOOTH}} &= \text{some even number}
\end{align*}
\]

"Look at the first equation," she said. "The left side tells how many boulders a strategy keeps for the 3-smooth case, and that must be an odd number to win, right?"

"Right," said Bob.

Alice continued. "The second equation tells us how many boulders we keep when I get a smooth stone, and you two are jagged. That must be an even number to win."

"I see," said Bob. "And the last two equations describe what must happen in the other possible 1 smooth cases."

"That’s right," said Alice. "If one of these equations fails, then the strategy fails for that case."

"Let me try building a strategy that satisfies all four equations," said Bob, and he created this table:

<table>
<thead>
<tr>
<th>Bob’s Sample Strategy</th>
<th>Smooth Numbers</th>
<th>Jagged Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice keeps every stone.</td>
<td>( A_{\text{SMOOTH}} = 1 )</td>
<td>( A_{\text{JAGGED}} = 1 )</td>
</tr>
<tr>
<td>Bob gives back every stone.</td>
<td>( B_{\text{SMOOTH}} = 0 )</td>
<td>( B_{\text{JAGGED}} = 0 )</td>
</tr>
<tr>
<td>Charlie gives back a smooth stone, but keeps a jagged stone.</td>
<td>( C_{\text{SMOOTH}} = 0 )</td>
<td>( C_{\text{JAGGED}} = 1 )</td>
</tr>
</tbody>
</table>

Alice said, "In your strategy, the fourth formula is odd, but it must be even to win the game. That’s why that particular strategy is no good."

"Ah ha!" said Charlie. "I see where you’re going. It’s impossible for any classical strategy to satisfy all the equations simultaneously, isn’t it?" Charlie added up the two sides of the four equations to show why:
BEATCS no 99

THE EATCS COLUMNS

\[
\begin{array}{c|c}
A_{\text{smooth}} + B_{\text{smooth}} + C_{\text{smooth}} & \text{some odd number} \\
A_{\text{smooth}} + B_{\text{jagged}} + C_{\text{jagged}} & \text{some even number} \\
A_{\text{jagged}} + B_{\text{smooth}} + C_{\text{jagged}} & \text{some even number} \\
A_{\text{jagged}} + B_{\text{jagged}} + C_{\text{smooth}} & \text{some even number} \\
\hline
2A_{\text{smooth}} + 2B_{\text{smooth}} + 2C_{\text{smooth}} & \text{some odd number} \\
+2A_{\text{jagged}} + 2B_{\text{jagged}} + 2C_{\text{jagged}} & \text{some odd number}
\end{array}
\]

"If all the equations worked, then the total on the left would be even, but the total on the right would be odd."

"You get full marks for that argument," Alice said. "For any classical deterministic strategy, those four equations can’t all be right, and that guarantees some failure."

They thought about putting randomness into the picture, perhaps throwing dice to determine whether to keep a stone. But once the dice were thrown, they would be back to a classical deterministic strategy that couldn’t guarantee success.

"That leaves us without a winning strategy," Charlie said.

"We need a strategy that wins 100% of the time," said Bob. "So we must reject all the classical strategies."

Telepathy Strategies

"I have an idea," said Bob, who had just been reading Robert A. Heinlein’s Time for the Stars. "We need to develop instantaneous telepathy."

"Sure," said Charlie. "I’m thinking of a number. What is it?"

Bob shut his eyes and screwed up his face. "Seven!" he said.

"No, but you’re close. It was 42,309,156."

After a few more attempts, they gave up on the idea, even though they all saw how telepathy would allow a winning strategy such as this:

- Alice and Bob always give back their boulders. Also, using instantaneous telepathy, they tell Charlie the shapes of their boulders.
- Now Charlie can figure out what kind of test they have. If they have a 3-smooth test, then Charlie keeps his boulder (so they have kept one-an

---

4If you’re still unconvinced, you can try all 64 of the classical deterministic strategies online at www.cs.colorado.edu/~main/qt/classical.html.

5... which is the way that Tom (on a starship) and Pat (on Earth) communicate in Time for the Stars.
od number); but if they have a 1 smooth test, then Charlie gives back his boulder (so they have kept zero—an even number).

Just to be sure that telepathy was the wrong path to pursue, Bob asked Dhaphod whether she could grant telepathy to them before the real test. Dhaphod denied the request, although she did produce some refreshments and a copy of a research paper called "Quantum Pseudo-Telepathy," written by Professors Gilles Brassard, Anne Broadbent and Alain Tapp of the Université de Montréal.

"This paper tells you how to win without telepathy,” she said.

The paper describes a truly amazing concept: A quantum computer that can perform feats which cannot be done in a world of classical physics. One of these amazing feats is to provide a non-classical strategy that always wins the boulder game. Keep in mind that the quantum computer used in the strategy does not have any faster-than-light transfer of information (no "telepathy"), but using quantum properties of superposition, interference and entanglement, it can guarantee that the Earthlings pass the test. It’s time for a quick diversion into the foundations of quantum computing.

Quantum Computers to the Rescue

A quantum computer manipulates pieces of information called qubits (pronounced cue-bits). The particular quantum computer that Alice, Bob and Charlie must build for the boulder game has three qubits, one for each Earthling. Each of these qubits has two possible values that we’ll call $\kappa$ and $\gamma$ (the Greek letters kappa and gamma). We could use 0 and 1 for the two possible values, but $\kappa$ and $\gamma$ have a nice mnemonic advantage that we’ll soon see.

Here’s how the strategy works: Alice, Bob and Charlie get together before anyone ever leaves Earth. At that time, they set up a three-qubit quantum computer in a particular state that we’ll describe in a moment. There are a variety of mechanisms that have been proposed for creating quantum computers like this, such as a mechanism that can trap a photon (or other particle) in one of two possible locations. The precise mechanism used doesn’t matter, so long as Alice, Bob and Charlie can each take one of the three qubits.

Once everyone arrives at their stars, the aliens give each Earthling a boulder. Alice, Bob and Charlie will each look at the shape of their boulder and choose one of two quantum computer programs (the SMOOTH program or the JAGGED program) to run on their qubit. After the programs run, Alice, Bob and Charlie each
look at their own qubits. In each case, if the qubit has the value of \( \kappa \), the Earthling keeps his or her boulder; if the value is \( \gamma \), the boulder is given back (so kappa means *keep* and gamma means *give back*). And the amazing part:

*The programs guarantee that the Earthlings will pass the test.*

Just how does this work? To answer that, we need the details that underlie quantum computation. We’ve already said that each qubit has two possible values: \( \kappa \) and \( \gamma \). But unlike ordinary pieces of information, a collection of qubits may be in a remarkable place called a superposition of observable states. When the values of such qubits are retrieved, the probabilities of observing different values are controlled by a mechanism called the quantum state of the computer.

For a quantum computer with three-qubits, the quantum state can always be represented using a short expression such as this one:

\[
0.7 |\kappa\kappa\kappa\rangle + 0.5 |\gamma\kappa\kappa\rangle + 0.1 |\gamma\kappa\gamma\rangle
\]

Each term in the expression has a number (called the amplitude of the term) and a triplet of possible qubit values (such as \( |\kappa\kappa\kappa\rangle \)). Each amplitude controls the probability of finding the quantum computer in a particular triplet. For example, \( 0.7 |\kappa\kappa\kappa\rangle \) means that there is a 49% probability of finding the computer in a state where all three qubits are \( \kappa \). The probability (49% in this case) comes from squaring the amplitude (\( 0.7 \times 0.7 = 0.49 \)). Overall, the quantum state \( 0.7 |\kappa\kappa\kappa\rangle + 0.5 |\gamma\kappa\kappa\rangle - 0.5 |\kappa\gamma\kappa\rangle + 0.1 |\gamma\kappa\gamma\rangle \) has:

- \( 0.7 \times 0.7 = 49\% \) chance of finding that all three bits are \( \kappa \) (which is \( |\kappa\kappa\kappa\rangle \)).
- \( 0.5 \times 0.5 = 25\% \) chance of finding \( |\gamma\kappa\gamma\rangle \), which means Alice has \( \kappa \) and the other two have \( \gamma \).
- \( -0.5 \times -0.5 = 25\% \) chance of finding \( |\gamma\kappa\kappa\rangle \), which means Alice has \( \gamma \) and the other two have \( \kappa \).
- \( 0.1 \times 0.1 = 1\% \) chance of finding \( |\gamma\kappa\gamma\rangle \), which means Bob has \( \kappa \) and the other two have \( \gamma \).

The states will always be arranged so that the total of all the probabilities is 100%.

**Interference**

Notice that one of the amplitudes, \(-0.5\), is negative. This is important because it leads to a pervasive quantum phenomenon called *interference*. To understand
interference, we can look at the simple example of water waves in the world of ordinary physics.

Think of a swimming pool. On the left end, Alice jumps in and creates a wave that moves away from the left edge of the pool. After a few seconds, the wave (traveling rightward) might look like this:

Suppose that shortly after Alice jumped, Bob also jumps in and creates his own wave that's also traveling rightward, but a little behind Alice's wave:

If Alice or Bob had jumped in alone, you would have seen one of the waves shown above. The wave is quite high at its crest. But if they both jump in, the two waves add together. Because they jumped at different times, one wave is high where the other is low, and vice versa. The high and the low pieces partly cancel out, to give a wave that has a much lower crest:

This situation of a high value (above the base line) and a low value (below the base line) canceling is called *destructive interference*.
Suppose we have a three-qubit quantum computer with an indeterminate starting state:

\[ 0.6 |\kappa\kappa\kappa\rangle + 0.8 |\gamma\gamma\gamma\rangle \]

If we examine the qubits at the start, there’s a 36% chance of finding three kappas and a 64% chance of finding three gammas. We then run a quantum computer program that converts the states. The \( \rightarrow \) symbol in the conversion means that the program converts the starting state on the left to the end state on the right:

\[ 0.6 |\kappa\kappa\kappa\rangle \rightarrow 0.36 |\kappa\gamma\gamma\rangle + 0.48 |\gamma\kappa\gamma\rangle \]
\[ 0.8 |\gamma\gamma\gamma\rangle \rightarrow 0.64 |\kappa\gamma\gamma\rangle - 0.48 |\gamma\kappa\gamma\rangle \]

It looks as if the final state could be either \( |\kappa\gamma\gamma\rangle \) (Alice has \( \kappa \) and the others have \( \gamma \)) or \( |\gamma\kappa\gamma\rangle \) (Bob has \( \kappa \) and the others have \( \gamma \)). But when we add the terms on the right side of the arrows, we get:

\[ 1.0 |\kappa\gamma\gamma\rangle + 0.0 |\gamma\kappa\gamma\rangle \]

The probability of the final state being \( |\kappa\gamma\gamma\rangle \) is 100%; a final state of \( |\gamma\kappa\gamma\rangle \) will never occur because of the destructive interference.

Figure 2: Interference When Two Quantum States Are Added

The same kind of destructive interference can occur in a quantum computer when two states combine. If the first state has a negative amplitude of -0.5 for the triplet \( |\gamma\kappa\kappa\rangle \), and the second state has a positive amplitude 0.5 for the same triplet, then in the combined state, the triplet \( |\gamma\kappa\kappa\rangle \) has an amplitude of zero, which means that \( |\gamma\kappa\kappa\rangle \) will never be found when the qubits are examined. A detailed example is shown in Figure 2, and here’s the key to the example:

There are certain triplets in the quantum computer where the sum of the amplitudes for that triplet is zero. That triplet will never be observed.

Destructive interference in the quantum computer is the same phenomenon that happens for waves in a pool. The only difference lies in what is being measured: In the pool, we measure the amplitude of a wave at a particular spot, whereas in the quantum computer we measure amplitude numbers attached to a particular triplet of qubits.

In writing a quantum computer program, the trick is to arrange for unwanted answers to have destructive interference that yields a 0% probability. For example, after Alice, Bob and Charlie are each given their boulders, we would like for them
to run quantum computer programs on their qubits to cause destructive interference that eliminates the combinations which fail the tests. That’s what all this destruction is aiming for, and we’ll get there soon, but we first need to describe more about the amplitudes that we attach to each triplet.

Amplitudes are sometimes simple decimal numbers such as 0.7 or −0.5, but it may be any complex number. For a triplet with a complex amplitude of $a + bi$, the probability of finding the triplet is given by $a^2 + b^2$ (the square of the magnitude of the complex number). For example, there are two amplitudes that we’ll often use, the real number $\sqrt{0.5}$, and the imaginary number $\sqrt{-0.5}$, both of which have a probability of 50%.

This interpretation of an amplitude was proposed by Max Born in 1926 because it matched the statistical outcome of experiments with quantum states, although Born viewed it as more than this. He regarded a probability wave as a real thing, more than a tool for statistical calculations. Born’s desire is still controversial (except perhaps to Dhaphod, who no doubt knows for sure), but it does have support from a mathematical formulation that interprets our "triplets" as basis vectors in a Hilbert space. Today, some physicists believe that this Hilbert space formulation follows from some very simple assumptions about the natural world, and moreover, within a Hilbert space, Born’s probability rule follows immediately from a theorem of Andrew Gleason. In other words, the probability interpretation is real because it is the only interpretation we could have reached from some simple assumptions.

**The Quantum Computer for the Boulder Test**

The research paper that Dhaphod gave to the students describes how to win a game that generalizes the boulder test. According to the paper, they can always pass the test by starting with a three-qubit quantum computer that is in the superposition state:

$$\sqrt{0.5} |\kappa\kappa\kappa\rangle + \sqrt{0.5} |\gamma\gamma\gamma\rangle$$

If they were to immediately look at their qubits, they would either see three kappas (50% probability) or three gammas (50% probability), but never any other combination.

Of course, they will not examine their qubits right away. Doing so would collapse the starting state to just one possibility, which would defeat the purpose. So, instead of immediately looking at the qubits, they each put their qubit in a safe place and head for the stars. Once they get to their destinations, they will each have two quantum computer programs that they can run on their own qubit before
looking at it. The two programs are called the SMOOTH program (which is run when the alien provides a smooth boulder) and the JAGGED program (which is run when the alien provides a jagged boulder). Each of these programs changes the qubits in a way that’s described by the formulae in Figure 3.

Alice, Bob and Charlie understood how everything would work, but even so, they asked Dhaphod whether she might allow a trial run. She kindly assented and showed them how to create three qubits. Then the Earthlings were taken to their faraway stars and presented with three boulders. In the trial run, all three boulders were smooth, so let’s see what happened in this case. As you read through the example, try to spot the point where destructive interference causes the unwanted combinations to vanish—that’s the magic that makes the quantum strategy do what no classical strategy can manage.

The three qubits started in the entangled state that we’ve already seen:

$$\sqrt{0.5}|\kappa\kappa\kappa\rangle + \sqrt{0.5}|\gamma\gamma\gamma\rangle$$

Everyone went off to their stars and three smooth boulders were handed out. Of course, none of the players knew this: Each knew only that his or her one boulder was smooth.

As it happened, Alice ran her program first. Figure 3 states that when Alice’s SMOOTH program runs, any term of the form $a|kyz\rangle$ will convert to two terms:

$$(\sqrt{0.5} \times a)|kyz\rangle + (\sqrt{0.5} \times a)|\gamma\gamma\rangle.$$ You can see that the first term in the start state, $\sqrt{0.5}|\kappa\kappa\kappa\rangle$, matches the form $a|kyz\rangle$ by setting $a$ to $\sqrt{0.5}$ and setting both $y$ and $z$ to $\kappa$; this means that when Alice ran her SMOOTH program, the term ($\sqrt{0.5}|\kappa\kappa\kappa\rangle$) was converted to ($\sqrt{0.5} \times \sqrt{0.5})|\kappa\kappa\kappa\rangle + (\sqrt{0.5} \times \sqrt{0.5})|\gamma\kappa\kappa\rangle$. In a similar way, Alice’s SMOOTH program also converted the second term in the start state to ($\sqrt{0.5} \times \sqrt{0.5})|\kappa\gamma\gamma\rangle - (\sqrt{0.5} \times \sqrt{0.5})|\gamma\gamma\gamma\rangle$. Overall, Alice’s SMOOTH program converted the start state to:

$$(\sqrt{0.5} \times \sqrt{0.5})|\kappa\kappa\kappa\rangle + (\sqrt{0.5} \times \sqrt{0.5})|\gamma\kappa\kappa\rangle + (\sqrt{0.5} \times \sqrt{0.5})|\kappa\gamma\gamma\rangle - (\sqrt{0.5} \times \sqrt{0.5})|\gamma\gamma\gamma\rangle$$

The product ($\sqrt{0.5} \times \sqrt{0.5})$ is exactly 0.5, so we can simplify the expression to:

$$0.5|\kappa\kappa\kappa\rangle + 0.5|\gamma\kappa\kappa\rangle + 0.5|\kappa\gamma\gamma\rangle - 0.5|\gamma\gamma\gamma\rangle$$

The formulae are examples of unitary transformations, which are the programming language for quantum computing algorithms. In addition, the entire program (that acts on a complete triplet) is formed as the tensor product of unitary transformations that act on individual qubits. For those with sufficient math background, these transformations are expressed in terms of linear algebra at www.cs.colorado.edu/~main/qt/linear.html.
These formulae tell how each of the quantum programs convert a state of the form $a|xyz\rangle$ to two new terms. Note that $a$ is the amplitude and $|xyz\rangle$ is the triplet of the term before running the program.

**How a term $a|xyz\rangle$ changes when**

A1. Alice runs the SMOOTH program on her qubit:

If $x = \kappa$, then $a|xyz\rangle$ is converted to $(\sqrt{0.5} \times a)|xy\zeta\rangle + (\sqrt{0.5} \times a)|yz\zeta\rangle$

If $x = \gamma$, then $a|xyz\rangle$ is converted to $(\sqrt{0.5} \times a)|xy\zeta\rangle - (\sqrt{0.5} \times a)|yz\zeta\rangle$

A2. Alice runs the JAGGED program on her qubit:

If $x = \kappa$, then $a|xyz\rangle$ is converted to $(\sqrt{0.5} \times a)|xy\zeta\rangle + (\sqrt{0.5} \times a)|yz\zeta\rangle$

If $x = \gamma$, then $a|xyz\rangle$ is converted to $(\sqrt{-0.5} \times a)|xy\zeta\rangle - (\sqrt{-0.5} \times a)|yz\zeta\rangle$

B1. Bob runs the SMOOTH program on his qubit:

If $y = \kappa$, then $a|xyz\rangle$ is converted to $(\sqrt{0.5} \times a)|x\zeta y\rangle + (\sqrt{0.5} \times a)|x\zeta z\rangle$

If $y = \gamma$, then $a|xyz\rangle$ is converted to $(\sqrt{0.5} \times a)|x\zeta y\rangle - (\sqrt{0.5} \times a)|x\zeta z\rangle$

B2. Bob runs the JAGGED program on his qubit:

If $y = \kappa$, then $a|xyz\rangle$ is converted to $(\sqrt{0.5} \times a)|x\zeta y\rangle + (\sqrt{0.5} \times a)|x\zeta z\rangle$

If $y = \gamma$, then $a|xyz\rangle$ is converted to $(\sqrt{-0.5} \times a)|x\zeta y\rangle - (\sqrt{-0.5} \times a)|x\zeta z\rangle$

C1. Charlie runs the SMOOTH program on his qubit:

If $z = \kappa$, then $a|xyz\rangle$ is converted to $(\sqrt{0.5} \times a)|x\zeta y\rangle + (\sqrt{0.5} \times a)|x\zeta y\rangle$

If $z = \gamma$, then $a|xyz\rangle$ is converted to $(\sqrt{0.5} \times a)|x\zeta y\rangle - (\sqrt{0.5} \times a)|x\zeta y\rangle$

C2. when Charlie runs the JAGGED program on his qubit:

If $z = \kappa$, then $a|xyz\rangle$ is converted to $(\sqrt{0.5} \times a)|x\zeta y\rangle + (\sqrt{0.5} \times a)|x\zeta y\rangle$

If $z = \gamma$, then $a|xyz\rangle$ is converted to $(\sqrt{-0.5} \times a)|x\zeta y\rangle - (\sqrt{-0.5} \times a)|x\zeta y\rangle$

Figure 3: How Alice, Bob and Charlie’s Programs Convert the Quantum State
They continued this process, running Bob’s and Charlie’s programs, which resulted in sixteen terms in the quantum state:

\[
0.25|\kappa\kappa\kappa\rangle + 0.25|\kappa\kappa\gamma\rangle + 0.25|\kappa\gamma\kappa\rangle - 0.25|\kappa\gamma\gamma\rangle \\
+0.25|\kappa\gamma\kappa\rangle + 0.25|\kappa\gamma\gamma\rangle - 0.25|\kappa\gamma\kappa\rangle + 0.25|\kappa\gamma\gamma\rangle \\
+0.25|\gamma\kappa\kappa\rangle + 0.25|\gamma\kappa\gamma\rangle - 0.25|\gamma\kappa\kappa\rangle + 0.25|\gamma\kappa\gamma\rangle \\
+0.25|\gamma\gamma\kappa\rangle + 0.25|\gamma\gamma\gamma\rangle + 0.25|\gamma\gamma\kappa\rangle - 0.25|\gamma\gamma\gamma\rangle
\]

In these sixteen terms each triplet occurred twice, and any two terms with the same triplet can be combined. For example, there are two terms equal to 0.25|\gamma\gamma\kappa\rangle, and when these are combined, the result is 0.5|\gamma\kappa\rangle. We also have two terms +0.25|\gamma\gamma\rangle and −0.25|\gamma\gamma\rangle, which cancel each other out when they are combined, so our final answer has no |\gamma\gamma\gamma\rangle term. Destructive interference got rid of the |\gamma\gamma\gamma\rangle term! After combining all possible terms, only four terms survive:

\[
0.5|\kappa\kappa\kappa\rangle + 0.5|\kappa\gamma\gamma\rangle + 0.5|\gamma\kappa\gamma\rangle + 0.5|\gamma\gamma\kappa\rangle
\]

Alice, Bob and Charlie now looked at their qubits. I’m not going to tell you exactly what they found, but you can see from the above quantum state that there were only four possibilities, and each of those four kept an odd number of boulders-so, they passed the test.

As an exercise, you might work through what happens for one of the 1-smooth tests. You can test your answer at www.cs.colorado.edu/~main/qt/quantum.html. You’ll find that the alien’s test is always passed—hoorah!—and you’ll see that the order in which Alice, Bob and Charlie run their programs does not affect the result.

**Entanglement and Beyond**

Just how did the quantum computer provide a guaranteed win? The key was designing the quantum program to produce destructive interference for the triplets that must be avoided. The program creates certain correlations between the individual components of the triplet, but the most remarkable aspect of these correlations is that they cannot arise in a world of classical physics. This is called quantum entanglement. It is the situation where individual measurable components of a system are linked together in a way that an individual component and its evolution over time cannot be described independently of the entire system—even when the components are widely separated.
The computations have another interesting aspect: After the test has finished, Alice, Bob and Charlie know that they have saved the human race, but they still don’t know the shapes of each others’ boulders. The quantum computation provides a guarantee that the test will be passed, but it does so without any exchange of information about the boulders’ shapes—and, in fact, no messages can ever be exchanged through these kinds of quantum experiments.

There is no real telepathy in our quantum computation, but even so, the physicists and computer scientists who designed these programs call the phenomenon quantum pseudo-telepathy, because from a classical point of view, it appears that our Earthlings must have engaged in some kind of instantaneous telepathy. That is where the primary interest lies in these kinds of quantum phenomena: In the world of quantum physics, it’s possible to solve problems that simply cannot be solved with classical physics. We can’t use quantum phenomena to build an instantaneous communicator, but we can solve puzzles that cannot be solved in the world of classical physics, and we thereby gain an understanding of the way the universe works.

So, we’ve passed a trial run. Are we ready to put Mother Earth on the line in a real test from the aliens? Not quite. Our current ability to create qubits and maintain their interconnectedness is limited to tiny time periods. In addition, physical implementations of the quantum algorithms using “quantum gates” are still largely on the drawing board. Not even Dhaphod would tell me how long we have to prepare for the real test; perhaps she didn’t know herself. But, in any case, it seems we have two choices: develop the hardware that’s needed for quantum computing—or start testing every newborn baby for instantaneous telepathy.

For Further Reading

Although mathematical, the "Quantum Telepathy" paper by Gilles Brassard, Anne Broadbent and Alain Tapp is worth looking at. It’s online at arxiv.org/abs/quant-ph/0407221.

The compendium Quantum Computation and Quantum Information by Michael A. Nielsen and Isaac L. Chuang is a great starting point for those who want complete information about quantum computing.

"The Limits of Quantum Computers" by Scott Aaronson (in Scientific American, March 2008, pp. 62–69) outlines which kinds of problems are good and bad candidates for solving by quantum computer. Professor Aaronson also has a widely followed blog at www.scottaaronson.com/blog.
"Quantum Computing with Ions" by Christopher R. Monroe and David J. Wineland (in *Scientific American*, August 2008, pp. 64-79) describes one of the most promising approaches to building the hardware of quantum computers.

**Acknowledgment**

I give my deep thanks to Professor G. Rozenberg and an anonymous referee for their encouragement and excellent suggestions on the specific kind of example to give for entanglement as well as many other aspects of the paper. Many thanks also to M. Serna for her friendly help in formatting the final paper.
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.

85 What’s next ?

A simple rule leads to the following sequence :

\[ u_0 = 4, u_1 = 3, u_2 = 3, u_3 = 5, u_4 = 4, u_5 = 4, u_6 = 3, u_7 = 5, u_8 = 5, u_9 = 4, \]
\[ \ldots, u_{100} = 10, \ldots, u_{999} = 24, u_{1000} = 11, \ldots \]

What is \( u_{10} \) ?

SOLUTIONS TO PREVIOUS PUZZLES

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?

**Solution** No.

Consider the set $A = \{(l, i, j, c) | 1 \leq l \leq 5, 1 \leq i < j \leq 5, \text{Both } (l, i) \text{ and } (l, j) \text{ are colored in } c\}$.

There are 20 different 3-tuples $(i, j, c)$ with $i < j$, thus if cardinal of $A$ is 21 or more, there are some $l_1 \neq l_2, i < j$ and $c$ such that $(l_1, i, j, c)$ and $(l_2, i, j, c)$ are in $A$ and thus you have a rectangle.

Thus, the set $A$ for a rectangle-free grid must be of cardinal 20 at most.

How many element in $A$ are there for a given line $l$? If the line is monochrome, there are $5 \times 4/2 = 10$ elements in $A$ from that line. If there is one box of a color and 4 of the other one, there are $4 \times 3/2 = 6$ elements in $A$ from that line. If there is 2 boxes of a color and 3 of the other one, there are $3 \times 2/2 + 2 \times 1/2 = 4$ elements in $A$ from that line. Thus, there are always at least 4 elements from each line.

Since there are 5 lines, cardinal of $A$ is at least $5 \times 4 = 20$.

Thus, the set $A$ for a rectangle-free grid must be of cardinal 20 exactly, with 4 elements from each line.

Thus each line $l$ will contain 3 boxes in a given color $c_l$ and 2 in the other one.

Assume now that there are 3 lines such that $c_l$ is white and 2 lines such that $c_l$ is black (the other cases are left to the reader). The number of elements in $A$ with $c = \text{white}$ is $3 \times (3 \times 2/2) + 2 \times (2 \times 1/2) = 11$, but there are only 10 different $(i, j)$ with $i < j$, thus there must exist $l_1 \neq l_2, i < j$ such that $(l_1, i, j, \text{white})$ and $(l_2, i, j, \text{white})$ are in $A$, that is you have a white rectangle.
REPORTS FROM

CONFERENCES
**Report on ICALP 2009**

The 36th International Colloquium on Automata, Languages and Programming. July 6–11, 2009, Rodos Island, Greece

Manfred Kudlek

ICALP 2009, the 36th in this series of conferences in Theoretical Computer Science, took place from July 6-10, 2009, including workshops from July 6-11, 2009, on Rhodos, the island of roses. Conference site was Rodos Palace Resort Hotel, with a somehow older style interior, situated at Ixia, about 5 km from the town centre of Rhodos City, at the beach of Ionian Sea.

It was organized by the ΕΡΕΥΝΗΤΙΚΟ ΑΚΑΔΗΜΑΪΚΟ ΙΝΣΤΙΤΥΤΟ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ (Research Academic Computer Technology Institute, (RA)CTI), and ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ (Department of Informatics and Telecommunications, National & Kapodistrian University of Athens), under the auspices of EATCS. The organizing staff consisted of Ioannis Chatzigiannakis, Efi Chita, Christina Dimopoulou, Rozina Efstrathiadou, Lena Gourdoupi, Elias Koutsoupias, Christos Kaklamanis, Sotiris Nikoletseas, Paul Spirakis, and Christos Zaroliagis.

ICALP 2009 was sponsored by the institutions CTI (Computer Technology Institute), ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩN (Ministry of Education & Religious Affairs), ΕΘΝΙΚΟ & ΚΑΠΟΔΙΣΤΡΙΑΚΩΝ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ, (National & Kapodistrian University of Athens), PIRAEUS BANK, AEOLUS, RIMAGO, Papasotiriou Books and more, and Google.

The colloquium was attended by 291 participants from 29 countries. Details are given in the following table:

| AT 1 | CN 1 | FR 31 | IN 3 | NO 8 | SG 1 | BE 3 | CZ 6 | GR 22 | IS 3 | NZ 1 | SK 1 | BR 1 | DE 23 | HK 2 | IT 36 | PL 5 | UK 15 | CA 16 | DK 4 | HU 2 | JP 6 | RU 2 | US 51 | CH 10 | ES 8 | IL 13 | NL 8 | SE 8 |

The 4 workshops were attended by 87 (the number may vary due to insufficient information) participants, all of whom were also attending ICALP. Exact figures are given below.

The scientific program consisted of 5 invited lectures, 107 presentations selected from 364 submissions (1 accepted paper was withdrawn after acceptance, 9 others were withdrawn) from 45 countries. 5 special lectures were presented in the honouring session for Christos Papadimitriou, and furthermore there was also
an EATCS award lecture. Details on the number of submissions, accepted papers by tracks A, B, C and countries, as well as by number of authors, are given in the tables below, where I, E, (A,B,C)S (A,B,C)A. stand for invited, special lecture, submitted and accepted papers in track A,B,C, and total submitted and accepted papers, respectively.

The program, as well as those of the workshops, can be found at http://icalp09.cti.gr/.

ICALP 2009 covered the following fields:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randomness in Computation</td>
<td>Logic and Formal Languages</td>
</tr>
<tr>
<td>Matching and Clusterings</td>
<td>Automata Theory, Algebraic Methods</td>
</tr>
<tr>
<td>Scheduling</td>
<td>Rewriting, Types, and Linear Logic</td>
</tr>
<tr>
<td>Data Structures and Algorithms for Large Data Sets</td>
<td>Games and Bisimulation</td>
</tr>
<tr>
<td>Graphs and Steiner Trees</td>
<td>Temporal Logics and Timed Automata</td>
</tr>
<tr>
<td>Communications and Communication Complexity</td>
<td>Approaches to Verification</td>
</tr>
<tr>
<td>Data Structures</td>
<td>Foundations of Distributed Computing</td>
</tr>
<tr>
<td>Sorting and Selection</td>
<td>Networks</td>
</tr>
<tr>
<td>Computational Complexity and Algebra Boolean Functions</td>
<td>Games and Markets</td>
</tr>
<tr>
<td>Tiling and Self-Assembly</td>
<td>AC</td>
</tr>
<tr>
<td>Data Streams</td>
<td>Data Structures and Security</td>
</tr>
<tr>
<td>Graphs</td>
<td>Wireless Networks</td>
</tr>
<tr>
<td>Approximation and Online Algorithms</td>
<td>Special Event</td>
</tr>
<tr>
<td>Fixed Parameter Algorithms and Complexity</td>
<td>Christos Papadimitriou</td>
</tr>
</tbody>
</table>

\[
\begin{array}{cccccccc}
\text{A} & \text{S} & \text{A} & \text{B} & \text{S} & \text{A} & \text{C} & \text{S} & \text{A} \\
1 & 46 & 8 & 23 & (3) & 4 & 1 & 73 & (12) & 13 \\
2 & 71 & 24 & 39 & 14 & 20 & 8 & 130 & 46 \\
3 & 60 & 19 & 17 & 5 & 23 & 8 & 100 & 32 \\
4 & 28 & 7 & 4 & 1 & 11 & 4 & 43 & 12 \\
5 & 11 & 3 & & & 2 & 1 & 13 & 4 \\
6 & 3 & 1 & 1 & & 1 & 5 & 1 \\
\hline
219 & 62 & 84 & (23) & 24 & 61 & 22 & 364 & (107) & 108 \\
\text{W} & 9 & 2 & & & 2 & & 13 \\
\end{array}
\]

Distribution by number of authors.
The Bulletin of the EATCS

<table>
<thead>
<tr>
<th>C</th>
<th>I</th>
<th>E</th>
<th>AS</th>
<th>AA</th>
<th>BS</th>
<th>BA</th>
<th>CS</th>
<th>CA</th>
<th>S</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT</td>
<td>1/3</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>3/5</td>
<td>1/5</td>
<td>7/12</td>
<td>1/7</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BE</td>
<td>1/3</td>
<td>1/3</td>
<td>1/7</td>
<td>2/3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BG</td>
<td>1/3</td>
<td>1/3</td>
<td>1/7</td>
<td>2/3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CA</td>
<td>13</td>
<td>23</td>
<td>43</td>
<td>1/3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CH</td>
<td>13</td>
<td>23</td>
<td>43</td>
<td>1/3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CL</td>
<td>1/3</td>
<td>1/3</td>
<td>1/7</td>
<td>2/3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CN</td>
<td>4/5</td>
<td>1/3</td>
<td>1/7</td>
<td>2/3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CY</td>
<td>1/3</td>
<td>1/3</td>
<td>1/7</td>
<td>2/3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CZ</td>
<td>13</td>
<td>23</td>
<td>43</td>
<td>1/3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DE</td>
<td>13</td>
<td>23</td>
<td>43</td>
<td>1/3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DK</td>
<td>1/3</td>
<td>1/3</td>
<td>1/7</td>
<td>2/3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EG</td>
<td>1/3</td>
<td>1/3</td>
<td>1/7</td>
<td>2/3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ES</td>
<td>1/3</td>
<td>1/3</td>
<td>1/7</td>
<td>2/3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FI</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FR</td>
<td>15</td>
<td>23</td>
<td>43</td>
<td>1/3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>GR</td>
<td>13</td>
<td>23</td>
<td>43</td>
<td>1/3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>HK</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>HR</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>HU</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>IE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IL</td>
<td>15</td>
<td>23</td>
<td>43</td>
<td>1/3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IN</td>
<td>13</td>
<td>23</td>
<td>43</td>
<td>1/3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IR</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IS</td>
<td>13</td>
<td>23</td>
<td>43</td>
<td>1/3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>IT</td>
<td>13</td>
<td>23</td>
<td>43</td>
<td>1/3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>JO</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>JP</td>
<td>8</td>
<td>13</td>
<td>23</td>
<td>1/3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LV</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NL</td>
<td>4</td>
<td>13</td>
<td>23</td>
<td>1/3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NZ</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NO</td>
<td>7</td>
<td>13</td>
<td>23</td>
<td>1/3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PL</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PT</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

175
ICALP 2009 was accompanied by the following 4 workshops:

**ALGOSENSORS 2009** (Fifth International Workshop on Algorithmic Aspects of Wireless Sensor Networks)

**DCM 2009** (5th International Workshop on Developments in Computational Methods)

**FOCLASA 2009** (8th International Workshop on Foundations of Coordination Languages and Software Architectures)

**QUANTLOG 2009** (Workshop on Quantitative Logics 2009)

Details on date (D), number of invited lectures (I), presentations (A), participants (P), and lecture room (R), are given in the following table:

<table>
<thead>
<tr>
<th>W</th>
<th>D</th>
<th>I</th>
<th>A</th>
<th>P</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>RO</td>
<td>1</td>
<td>3</td>
<td></td>
<td>1</td>
<td>1/12</td>
</tr>
<tr>
<td>RU</td>
<td>2/12</td>
<td>1/5</td>
<td></td>
<td>1/5</td>
<td>3/12</td>
</tr>
<tr>
<td>SE</td>
<td>3/4</td>
<td>1</td>
<td>7/9</td>
<td>3/2</td>
<td>2/3</td>
</tr>
<tr>
<td>SG</td>
<td>2/9</td>
<td>1/5</td>
<td></td>
<td>7/9</td>
<td>1/5</td>
</tr>
<tr>
<td>SK</td>
<td>1/3</td>
<td>1</td>
<td></td>
<td>1/3</td>
<td>1/3</td>
</tr>
<tr>
<td>TR</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>TW</td>
<td>1/3</td>
<td></td>
<td>1/3</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td>US</td>
<td>3/7</td>
<td>1</td>
<td></td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td>UK</td>
<td>1/3</td>
<td></td>
<td>3/10</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>5/3</td>
<td>1</td>
<td>3/10</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
<tr>
<td></td>
<td>7/3</td>
<td></td>
<td>3/12</td>
<td></td>
<td>1/3</td>
</tr>
</tbody>
</table>

The presentations were given in 28 sessions, organized in 2 parallel sessions, 16 in track A (A10 and A11) in parallel, 6 in B, 4 in C, and 2 in combined AC. They were held in rooms Salon des Roses and Nefteli.

All contributions were presented, although that of Adam R. Klivans, Philip M. Long, Rocco A. Servedio by Ryan O'Donnell, and that of Endre Boros, Kazushisa Makino by Magnús M. Halldórsson.

The conference was opened on Monday morning by Paul Spirakis, welcoming all participants. Then

Kurt Mehlhorn (co-authors Naveen Garg, Amit Kumar, Julian Mestre, Kavitha Telikepalli) gave a very good and interesting first invited lecture with ‘Assigning Papers to Referees’ in which he presented a bipartite graph with weights as mathematical model of the problem, assignment as subset of edges. He also
discussed quality of reviewers, fairness, signature, ideas from social sciences, the NP-completeness of the problem, and difficulties with existing real systems as Easychair. Unfortunately, there is only a short abstract in the proceedings, however the paper is available at http://www.mpi-inf.mpg.de/~mehlhorn/ftp/RefereeAssignment.pdf.

Thomas A. Henzinger (co-authors Krishnendu Chatterji, Laurent Doyen), with the second one, ‘A Survey of Stochastic Games with Limsup and Liminf Objectives’, gave another very good and interesting presentation on reactive systems and graph games, more general than in the proceedings, with 1, 1 \( \frac{1}{2} \), 2, ..., players, on their strategy and computational complexity, symbolic fixed point, representation in modal logic, and automata and operators related to it.

The third, very good and interesting, fast invited talk was given by Georg Gottlob (co-authors Gianluigi Greco, Francesco Scarcello) with ‘Tractable Optimization Problems through Hypergraph-based Structural Restrictions’. It was a nice overview on the constraint satisfiability problem, Boolean conjunctive query evaluation, with examples of combinatorial crosswords puzzle, robber-cop and robber-cop-marshall game, using hypergraphs for description.

Noam Nisan (co-authors Jason Bayer, Deepak Chandra, Tal Franji, Robert Gardner, Yossi Matias, Neil Rhodes, Misha Seltzer, Danny Tom, Hal Varian, Dan Zigmond), with the fourth one, ‘Google’s Auction for TV Ads’, presented a good and interesting survey on Google’s activities in selling ads (not on economic and social aspects, why doing it, future plans, how much money), in particular the auction system, what was done, not on numbers (‘some I don’t know, some I can’t tell’), the basic theoretical model, the auction goals, ascending auction (good and bad), and the ‘ugly’.

A highlight was the excellent fifth, well illustrated, invited lecture ‘Algorithmic Game Theory: A Snapshot’ by Christos H. Papadimitriou. In the first part he presented a survey on the history of algorithmic game theory, introduced by John von Neumann before 1954, starting with ‘Don’t expect to learn much, just my last results’. He then talked on Nash equilibrium, non-effective algorithms to find it, its importance (rationality, model of behaviour), its intractability (PPAD-complete) to find unbalanced nodes, the price to find the equilibrium, exponential lower bounds, algorithmic mechanism design, and the price of anarchy. The second part he started with ‘And now’ (last day of ICALP and last invited talk) ‘for something entirely different! Sex!!!’, pausing shortly, looking for reaction, and then talking about (dis)advantages of sex (ubiquitous, very costly) on simulated annealing algorithms (asexual reproduction, working), genetic algorithms (sexual reproduction, not working), mixability, quality of being on plateau and not on peak, sex as origin of genes and genetic modularity, and thus Darwin’s principle survival of the fittest’ does not hold.
A special session was held on Wednesday afternoon honouring the contributions of Christos Hariláou Papadimitríou on Theoretical Computer Science, and also his birthday (which seems to be a secret). It was opened by Elias Koutsoupias, talking on the scientific life of the honoured, his teaching, books, awards, positions, and himself being perhaps the most difficult student of Christos. Paul G. Spirakis then announced a special surprise, a special issue of Computer Science Review (Volume 3, Issue 2), edited by Mavros Mavronicolaos and himself, dedicated to honour the personality and scientific life of Christos. All participants got a copy.

It was followed by five excellent and interesting special talks.

Richard Karp gave the first one with ‘Three Combinatorial Problems in Genomics and Computational Biology’. It was a survey on the discovery of protein molecules, associations between them, the cellular machinery, conserved protein interaction networks, topology-free alignment, algorithms and their complexity, genomic variation, multi-game alignment, complexity bounds, and graph-theoretic formulation.

The second one, ‘Randomly Grown Graph Sequences: Spectra and Limits’ was presented by László Lovász, starting with ‘Happy Birthday Christos!’. He presented an overview on large networks as part of many sciences, challenge for mathematics and computer science, Erdős-Rényi and Albert-Barabási graphs, degree distribution, density and degree bound, and limits of graph series, also showing a number of examples., and once more congatulating Christos.

Noam Nisan (co-authors Shahar Dobzinski, Ron Lavi) gave the third one, ‘Multi-unit Auctions with Budget Limits’, in which he gave a survey on auction and a quasi-linear model, target, bid and budget in ad auctions, the revelation principle, the non-existence of incentive-compatible Pareto-optimal auctions, and the market equilibrium.

In the fourth, fast talk ‘Papadimitriou and the Price of Anarchy’ (not a political talk!), Tim Roughgarden, starting with a story of some UCB lunch in 1999, presented an overview on the quality of an equilibrium, an intrinsic definition of the price of anarchy, Nash and other equilibria and their relation to each other, smoothness bounds, and tight game classes.

Mihalis Yannakakis, in the fifth one, ‘Facts of Complexity’, started with ‘I’ll speak a little bit more time’ and ‘Εν τού ἔργῳ ὑπό τὸ ὄνομα τούτου...’ (In the beginning there was ...). He then gave a survey on the history of complexity from 1975 up to now, related to Christos, as travelling salesman problem, approximation, local search, neural networks, congestion games, PPAD completeness, irrationality, Euclidean TSP, fixed point problems, price adjustment methods, and relations between complexity classes. He finished with ‘Thanks Christos! (for 35 years).

Finally, Christos Papadimitriou thanked all colleagues, friends, and the community, in particular the speakers of this special event. Then he also mentioned...
his many students, especially Paris Christos Kanellakis, who passed away too young with his family in an airplane accident at Cali just before Christmas in 1995, Elias Koutsoupias, Paul Spirakis (who was not his student), and last not least Mihalis Yannakakis whom he had tried very hard to convert to computer science.

On Thursday afternoon Girogio Ausiello introduced, and Catuscia Palamidessi explained the EATCS award and the decision to honour Gérard Huet with it, and introduced his scientific life. Gérard Huet, he had birthday 2 days before, then gave a nice talk which may be titled ‘Diversity and Fun in Research’, starting with ‘I am wondering why I deceive it, more software architect, theory only at demand’, and thanking all his colleagues and students. Then he talked on network and great variety of researchers, and anecdotes of his career, as AI when having a dream 40 years ago to go to USA, his master thesis in that field, data problems, operating systems, \( \lambda \) calculus, and the accident of a friend who drunk too much, with the solution ‘you should not drink, but should look around’ applied to solve mathematics and diversity. Later he was at IRIA, where was total freedom of research topics, meeting many researchers as Corrado Böhm, Robin Milner, Brian Mayoh. He also mentioned that he is indebted to former recipients of the award as Nicolas de Bruijn, and also to Peter Landin who passed away on June 3, 2009. Finally he spoke on the Sanskrit heritage project, in particular natural language processing, the clear structure of the system, and its relation to Eilenberg machines.


From the other contributions only some personal impressions can be given. Good and interesting talks were given by Mark Weyer on decidability of the boundedness problem for MSO logic over finite words, by Luke Ong on complexity of model checking recursion schemes for fragments of modal \( \mu \)-calculus, saying about a definition ‘this might not be very helpful, let me try it again’, by Konrad Zdanowski on a tight lower bound for translation of Büchi automata to deterministic Rabin automata, by Martin Beaudry on faithful loops for certain aperiodic ordered monoids, by Thomas Colcombet on regular cost functions as extensions of regular languages and related stabilization monoids, and by Jean-Pierre Jouannaud on diagrammatic confluence and completion in proof reduc-
Other good and nice talks were given by Lucia Acciai on decidability in infinite-state π-calculus, by Éric Ruppert on names trump malice, a new theoretical model of mobile computation, finishing with ‘It’s actually more complicated like this. See proceedings!’ by Vojtěch Forejt on reachability in stochastic timed games, with players Elpis, Ares and Tyche, by Philippe Chaput, speaking very fast, on approximating Markov processes, and by Tobias Friedrich on quasi-random rumor spreading, expansion and robustness.

Also to mention are the interesting and good presentations by Felix Klaedtke on regular temporal logic with past and its expressiveness, by Igor Walukiewicz on alternating timed automata, by Thomas Brihaye on determinization of timed automata and its complexity, by Jean Gibault-Larrecq on well structured transition systems and their completion, well illustrated, by Rahul Santhanam on lower complexity bounds for exponential time classes, and by Neeraj Kayal on polynomial time factorization of finite groups.

Good and interesting talks were given by Bruno Durand on aperiodic tiles with high Kolmogorov complexity, with colourful illustrations, by Tim Roughgarden on worst-case efficiency analysis of queueing disciplines, by Aggelos Kiayias on secure functions collection, showing nice illustrations as a red devil with fork, by Steve Chien on price of anarchy in congestion games, and by Nathalie Aubrun on finite type tree shifts in infinite trees and their decidability of congruacy.

Excellent presentations were given by Jean-Éric Pin on polynomial closure of a lattice of regular languages, by François le Gall on a general scheme for a perfect quantum network, and by John Reif on self-assembly in nature, its tile assembly model and complexity.

To mention are also Simone Martini dedicating his talk to the memory of Peter Landin, Francesco Tiezzi presenting a cow at a computer (he talked about the COWS system), Mario Szegedy, first searched for by the chairlady Anna Gál, and after some time saying ‘now my slides are permanently permuted’, and Constantinos Daskalakis finishing with Καλό Καλόκαιρι!.

Sometimes technical problems arose when uploading the talks to foreign laptops, or with connection to the beamer. It might have been better to achieve that before the sessions onto only one computer.

The proceedings, containing all invited talks, that of Kurt Mehlhorn only as abstract, and all contributions, edited by Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, have been published as Springer LNCS 5555 with track A, and LNCS 5556 with tracks B and C, in a new subseries Advanced Research in Computing and Software Science (ARCoSS).
In the breaks coffee, tea, juices, mineral water, and cakes were offered. Access to internet was only possible wireless and with own laptops near the lecture rooms.

There was also the traditional book exhibition of Springer, as well as of Cambridge and Oxford University Presses.

The social program started on Monday late afternoon with a welcome reception at the bar near the hotel swimming pool, with Sangria, juice, wine, beer, snacks and salad.

| Jean-Éric Pin | Kurt Mehlhorn | 11 7.
| Juhani Karhumäki | 9 3.
| Mihalis Yannakakis | 8 4.
| Zvi Galil | 8.
| Amir Pnueli | 7 2.
| Christos Papadimitriou | 7 3.
| Philippe Flajolet | 7 2.
| Grzegorz Rozenberg | 7.
| Paul Vitányi | 6 1.
| Claus-Peter Schnorr | 6 2.
| Torben Hagerup | 6 2.
| Géraud Sénizergues | 6 2.
| John Reif | 6 2.
| Karel Čulik II | 6.
| Walter Vogler | 6.
| Joost Engelfriet | 5 2.
| Matthew Hennessy | 5 2.
| Arto Salomaa | 5 2.
| Juris Hartmanis | 5 2.
| Andrzej Lingas | 5 2.
| Michael Rabin | 5 2.
| Thomas Henzinger | 5 2.
| Burkhard Monien | 5 2.
| Ronald Book | 5 2.
| Christian Choffrut | 5.
| Arnold Schönhage | 5.
| Leslie Valiant | 5.

**ICALP Contributors**

| Jean-Éric Pin | Kurt Mehlhorn | 11 7.
| Dominique Perrin | 4 2.
| Zohar Manna | 4 2.
| Bruno Courcelle | 4 2.
| Moshe Vardi | 4 2.
| Juraj Hromkovič | 4 2.
| Paul Spirakis | 4 2.
| Thomas Wilke | 4 2.
| Denis Thérien | 4 2.
| Manfred Droste | 4 2.
| Robin Milner | 4 2.
| Moti Yung | 4 2.
| Bernard Chazelle | 4 2.
| David Peleg | 4 2.
| Ming Li | 4 2.
| Amin Coja-Oghlan | 4 2.
| Maurice Nivat | 4 2.
| Volker Diekert | 4 2.
| Piotr Berman | 4 2.
| Marek Karpiński | 4 2.
| Sudipto Guha | 4 2.
| Christophe Reutenauer | 4.
| Marcel Paul Schützenberger | 4.
| Davide Sangiorgi | 4.
| Colin Stirling | 4.
| Thomas Colcombet | 4.
| Ian Munro | 4.

181
On Tuesday late afternoon the EATCS General assembly was held on which the winners of the best paper awards received their prices, the sites for next ICALP’s were presented, Bordeaux (2010) by Igor Walukiewicz, Zürich (2011) by Beat Gfeller, and Warwick (Coventry) (2012) by Artur Czumaj. The author of this report offered EATCS buttons to the editors of the ICALP 2009 proceedings. Nobody reached 5 full papers this time. The current state is given in the table above.

The banquet was held on Wednesday evening in the big half sphere hall of the hotel, with warm and cold buffet, wine, juice, mineral water, and coffee. On an island in a small lake in the centre a lady played music, more or less without Greek tunes. It lasted well until 23 h. There was no excursion this time, but that was compensated by the special session for Christos just before the dinner.

On Thursday late afternoon we had another party, again at the bar near the swimming pool, with drinks and snacks as at the welcome reception.

Most participants stayed at Rodos Palace hotel where also lunch was served. Weather was hot, blue sky without clouds, and with highest temperatures between 30 and 35 ° C.

ICALP 2009 was successful again, on a high scientific level, and well organized. Pictures of the event can be found at http://icalp09cti.gr/. Next ICALP, will be at Bordeaux, from July 6-10, 2010, with workshops from July from July 5-12, 2010.

Avdios Poosoc and Bienvenue à Bordeaux.
REPORT ON CPM 2009
20th Annual Symposium on Combinatorial Pattern Matching
June 22–24 in Lille, France

Travis Gagie

Stepping out of the Flandres station on Midsummer’s Day, it was clear at once
that extraordinary things were happening in Lille: the streets were full of musi-
cians engaged in an acoustic free-for-all, celebrating the solstice, and giant black
statues of babies with bats’ wings and dragons’ tails, part of the EuropeXXL ex-
hibit. After a short walk to soak up some of the festivities, I headed back to catch
the miraculously efficient driverless metro. A few minutes would have taken me
to the weekly Wazemmes market but I headed instead to the Cité Scientifique,
which for the next three days would host the event for which I had really come to
France: the 20th Annual Symposium on Combinatorial Pattern Matching, known
to its friends as CPM ’09.

The symposium showcased the latest advances not only in pattern matching
but also in compression, indexing, stringology and bioinformatics. It included
three invited talks, a retrospective, a birthday celebration and — naturellement!
— plenty of wonderful French food and drink. The first morning gave us a chance
to catch up with colleagues and sample gaufres fourrées (special Lille wa-
ffles) before listening to four regular talks on compression and indexing and a fasci-
nating invited talk by Roberto Grossi from the University of Pisa, who gave us a
whirlwind tour of “Text Indexing, Suffix Sorting and Data Compression: Com-
mon Problems and Techniques”. (More details of all the talks are available on
the conference website, http://bioinfo.lifl.fr/cpm09/doku.php, and in
the proceedings, LNCS Volume 5577; some will also appear in a special issue of
Theoretical Computer Science.) A delicious lunch in the campus restaurant’s VIP
room was followed by six more regular talks on compression, string algorithms
and pattern matching. The day ended with a banquet at the restaurant of the Palais
des Beaux Arts, during which the organizers used tables, charts, a coauthorship
graph and even a Wordle (all to be found in the proceedings) to illustrate statistics
about the symposium’s history. They also announced the winners of the Best Pa-
per Award and the Best Student Paper Award — Rolf Backofen, Dekel Tsur, Shay
Zakov and Michal Ziv-Ukelson for “Sparse RNA Folding: Time and Space Ef-
ficient Algorithms” and Djamal Belazzougui for “Faster and Space-Optimal Edit
Distance ’1’ Dictionary”, respectively — and presented the Festschrift London
Algorithmics 2008: Theory and Practice to Maxime Crochemore, the chair of the
first CPM, in honour of his sixtieth birthday.
The second day brought more regular talks, on approximate string matching (including Belazzougui’s paper), string algorithms and parameterized complexity, as well as a timely invited talk on “Modeling and Algorithmic Challenges in Online Social Networks” by Ravi Kumar of Yahoo! Research. The academic activities were followed by a guided tour of Old Lille, with its mix of French and Flemish influences, and a beer tasting (in which the Flemish influence was perhaps predominant). Although delectable hors-d’œuvres were served with the beer, some of us still found room afterwards for some Lille specialties, such as mussels and regional cheeses. The third and final day was devoted to bioinformatics (including Backofen, Tsur, Zakov and Ziv-Ukelson’s paper) except for the invited talk by Christos Faloutsos of Carnegie Mellon on “Graph Mining: Patterns, Generators and Tools”. The second and third invited talks together provided a cutting-edge survey of the challenges, techniques and potential rewards of studying massive networks. Everyone I spoke to agreed that this year’s CPM was a great success and that many thanks are due to the organizers and sponsors, in particular the co-chairs of the program committee, Gregory Kucherov and Esko Ukkonen; the co-chairs of the organizing committee, Professor Kucherov again and Hélène Touzet; and the student volunteers, who worked solely for their love of pattern matching (and perhaps the promise of Carambar candies).
REPORT ON DCFS 2009
The 11th Annual Workshop of Descriptional Complexity of Formal Systems
July 6–9, 2009, Magdeburg, Germany

Ian McQuillan
University of Saskatchewan

The Eleventh Annual International Workshop on Descriptional Complexity of Formal Systems (DCFS 2009) was held at Otto von Guericke University Magdeburg, in Magdeburg, Germany, on July 6–9, 2009. It was organized jointly by the IFIP Working Group 1.2 on Descriptional Complexity, and by the Faculty of Computer Science of the Otto von Guericke University Magdeburg. It took place immediately after the Conference on Developments in Language Theory (DLT), in Stuttgart, Germany.

Papers were submitted by 38 authors from 15 different countries. The Program Committee selected 16 papers for presentation. The program can be found on the conference website http://theo.cs.uni-magdeburg.de/dcfs2009/.

The conference was organized by Jürgen Dassow (chair), Ronny Harbich, Bernd Reichel, Bianca Truthe and Catharina Berner. The conference went forward smoothly, and the participants were able to engage in research discussion.

The organizers of DCFS 2009 were Ronny Harbich, Bianca Truthe, Jürgen Dassow (chair), Catharina Berner, Bernd Reichel (from left to right)
The proceedings, edited by Jürgen Dassow, Giovanni Pighizzini and Bianca Truthe, contain all invited talks and contributions, and have been published by Otto von Guericke University Magdeburg. They are also available in Electronic Proceedings in Theoretical Computer Science http://published.epjcs.org. Full versions of selected papers will be published in a special issue of the Journal of Automata, Languages and Combinatorics.

The invited lectures covered a broad range of topics in Descriptional Complexity. They were (in order of appearance),

- **Janusz Brzozowski** (Waterloo, Canada). “Quotient Complexity of Regular Languages.”

- **Christel Baier** (Dresden, Germany). “Probabilistic Automata over Infinite Words: Expressiveness, Efficiency, and Decidability.”

- **Nicole Schweikardt** (Frankfurt/Main, Germany). “The Descriptional Complexity of Fragments of Monadic Second-Order Logic.”


As part of the program, Jürgen Dassow gave an excellent presentation entitled Ten Years DCFS, which summarized many of the significant works from the history of the workshop. It was especially remarkable to see how the articles from DCFS have collectively advanced the field. It also offered future directions in descriptional complexity, and some interesting open questions to carry the workshop forward for another ten years. This presentation is available online at http://theo.cs.uni-magdeburg.de/dcfs2009/text/TenYearsDCFS.pdf. Giovanni Pighizzini also gave a presentation on important questions that can be addressed in the future. At the Business Meeting, a preview of the next DCFS was given by Ian McQuillan who will organize the workshop in Saskatoon, Canada in August 2010. DCFS 2010 will take place directly before CIAA and DLT, both also in Canada.

Our social excursions took place over two days. The first took the participants to the 800 year old Cathedral of Magdeburg. The next day we were taken on a scenic tour of some unique sights outside Magdeburg. We took a guided tour of Tangermünde, a 1000 year old town. Then, the participants visited a late Romanesque style monastery in Jerichow. There was also a visit to Europe’s longest aqueduct, where the Midland Canal crosses overtop the river Elbe. These social events culminated in the conference dinner which was held in nearby Hohenwarthe.
Many of the participants of DCFS 2009
Thank you to all the organizers of DCFS 2009, to the speakers for the interesting talks, and to the participants. We welcome you to DCFS 2010 in Saskatoon, Canada!
REPORT ON CSR 2009

The 4th International Computer Science Symposium in Russia
18–23 August 2009 in Novosibirsk, Russia

E.A. Hirsh

The International Computer Science Symposium in Russia (CSR, http://logic.pdmi.ras.ru/~csr/) is a series of international conferences united by their place (Russia) started with CSR-2006 in St.Petersburg, CSR-2007 in Ekaterinburg, and CSR-2008 in Moscow.

The 4th event in this series, CSR-2009, was held on 18–23 August 2009 in Novosibirsk, Russia, organized by Anna Frid and hosted by Sobolev Institute of Mathematics and Novosibirsk State University.

The opening lecture was given by Andrei Voronkov, and four other invited plenary lectures were given by Sergei Odintsov, Wolfgang Thomas, Nikolai Vereshchagin, and Hongseok Yang. The proceedings of the symposium were published in Springer’s Lecture Notes in Computer Science, number 5675.

The next symposium CSR-2010 will be held in Kazan in June 2010, and we are looking forward to see everyone again there (the deadline is December 7, 2010!).

Pictures from CSR 2009

Olaf Beyesdorff  Mikhail Vyalyi
Klaus Wagner  Yi Deng  Ely Porat
Edward Hirsch  Yuri Pritykin  Andrei Romashchenko
Andreas Goerdt  Chris Calabro  Victor Selivanov
Wolfgang Thomas  Tommy Färnqvist  Pit van ’t Hof
REPORTS FROM CONFERENCES

Galina Jirásková
Kristoffer Arnsfelt Hansen
Maurice Jansen

Andrey Morozov
Andrei Voronkov
Abuzer Yakaryılmaz

Denis Krotov
Alexander Smal
Sergey Nikolenko

Jayalal Sarma
Somnath Sikdar
Maurice Jansen
REPORT ON WG 2009
The 35th International Workshop on
Graph Theoretic Concepts in Computer Science
Hajo Broersma

From June 24-26 2009 the 35th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2009, was held in Montpellier in the South of France.

As usual with the WG meetings, post-proceedings will appear in the Lecture Notes in Computer Science series of Springer-Verlag.

The charming city of Montpellier is situated on hilly ground 10 kilometres inland from the Mediterranean coast on the River Lez. It is the capital of the Languedoc-Roussillon region, as well as the Hérault department. Although it is not quite on the Mediterranean itself it has a very Mediterranean feeling. Montpellier is a little over 1000 years old, and is on the old pilgrim trail to Santiago de Compostela, markers for which run through the town. There are many open squares where you can sit down and have a drink or meal in the city due to the monasteries and other buildings that were torn down during the French revolution. The conference took place in an old movie theater in the old part of town while most of the participants stayed in hotels in the new part. This made for very nice walks through the town by the participants who chose not to use the convenient blue tram link.

WG 2009 was the second such meeting in France, but continued the series of 34 previous ones. Since 1975, WG took place twenty-one times in Germany, four times in The Netherlands, twice in Austria and France, as well as once in the Czech Republic, England, Italy, Norway, Slovakia, and in Switzerland.

Out of 70 eligible submissions the international programme committee selected 28 for presentation at the workshop. In addition to these regular talks there were two excellent invited lectures. About 75 participants from all over the world attended WG 2009.

After the welcome, registration and wine and cheese party on Tuesday evening, the scientific program started on Wednesday morning with the first invited lecture by Daniel Král from Prague on graphs of bounded expansion. This is a notion of sparsity in graphs, based on shallow minors in which two vertices may only be contracted together if they started out within some constant distance of each other in the original graph. There is a trichotomy theorem for the density of shallow minors. Graphs of bounded expansion form the case in which the shallow minors are all sparse; they generalize both minor-closed graph families and bounded-degree...
graphs. As Král described in his excellent and very stimulating talk, graph properties describable as first-order logic formulae may be tested efficiently on these graphs. He made clear to the audience that there is plenty of opportunity for new work in this area.

The second invited talk was given after the lunch break on Thursday by David Eppstein. He gave a splendid lecture on graph-theoretic solutions to computational geometry problems. The general approach is that the geometry leads to an auxiliary graph and that the special properties of the auxiliary graph yield nice algorithmic solutions that can be translated back to the geometry problem. He sketched this approach to seven problems: art gallery theorems, partition into rectangles, minimum diameter clustering, bend minimization, mesh stripification, angle optimization of tilings, and metric embedding into stars. He showed himself a very good ambassador for WG-type research by convincing the audience that a graph-theoretic point of view is useful in many non-graph problems. The graph algorithms used for these problems are often classical but sometimes with a twist. Special classes of graphs and their structure are often important as well.

For the second time in the history of WG, the meeting in 2009 offered a best student paper award. On behalf of the WG steering committee, Martin Golumbic presented this award and the cheque of €500 in prize money to Ignasi Sau for his joint paper with Zhentao Li entitled graph partitioning and traffic grooming with bounded degree request graph.

All but one of the accepted papers were presented at the meeting by one of the authors; the other accepted paper was presented by a colleague of the authors. The talks showed a variety of topics concerning graph models, graph algorithms and their structural and computational complexity, with many of their aspects in relation to computer science and applications.

Overall the contributions were of a very high level and convinced the audience by the results disseminated and the quality of the presentations. The 28 contributed talks and the two invited lectures made an excellent scientific programme.

Before the Conference Dinner on Thursday evening, there was an interesting and educational excursion in which we learned some of the local history by visiting three local landmarks (the opera house, an old pharmacy, and the top of the gateway arch) and drinking a different wine in each.

The pleasant location, the excellent organization, the very interesting scientific programme and the stimulating atmosphere among the participants made this a very enjoyable meeting. For further information on WG 2009, including the full programme and photos, go to http://www.lirmm.fr/wg2009/. We are looking forward to WG 2010, which will take place on Crete in Greece from June 28-30.
HISTORY AND ORGANIZATION

EA TCS 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 EA TCS, which elects a President, Vice Presidents, and a Treasurer. Policy guidelines are determined by the Council and the General Assembly of EA TCS. This assembly is scheduled to take place during the annual International Colloquium on Automata, Languages and Programming (ICALP), the conference of EA TCS.

MAJOR ACTIVITIES OF EA TCS

- Organization of ICALP;
- Publication of the “Bulletin of the EA TCS;”
- Award of research and academic careers prizes, including the “EA TCS 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 EA TCS 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 EA TCS include:
- Subscription to the “Bulletin of the EA TCS;”
- Reduced registration fees at various conferences;
- Reciprocity agreements with other organizations;
- 25% discount when purchasing ICALP proceedings;
- 25% discount in purchasing books from “EA TCS Monographs” and “EA TCS 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, Demark 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
- Bologna, 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. 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](http://www.eatcs.org), or contact the President of EATCS:

**Prof. Dr. Giorgio Ausiello, Dipartimento di Informatica e Sistemistica
Universita 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](http://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](http://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*