ISSN 0252–9742

Bulletin

of the

European Association for

Theoretical Computer Science

EATCS

Number 119 June 2016

EATCS Council Members

email addresses

Luca Aceto	luca@ru.is
Lars Arge	large@madalgo.au.dk
Jos Baeten	jos.baeten@cwi.nl
Lars Birkedal	large@madalgo.au.dk
Mikolaj Bojanczyk	bojan@mimuw.edu.pl
Fedor Fomin	fomin@ii.uib.no
Pierre Fraigniaud	pierre.fraigniaud@liafa.univ-paris-diderot.fr
Leslie Ann Goldberg	leslie.goldberg@cs.ox.ac.uk
Magnus Halldorsson	magnusmh@gmail.com
Monika Henzinger	monika.henzinger@univie.ac.at
Kazuo Iwama	iwama@kuis.kyoto-u.ac.jp
Dirk Janssens	Dirk.Janssens@ua.ac.be
Christos Kaklamanis	kakl@ceid.upatras.gr
Antonin Kucera	tony@fi.muni.cz
Elvira Mayordomo	elvira@unizar.es
Michael Mitzenmacher	michaelm@eecs.harvard.edu
Anca Muscholl	anca@labri.fr
Luke Ong	luke.Ong@cs.ox.a.uk
Catuscia Palamidessi	catuscia@lix.polytechnique.fr
Giuseppe Persiano	giuper@dia.unisa.it
Alberto Policriti	alberto.policriti@uniud.it
Alberto Marchetti Spaccamela	alberto@dis.uniroma1.it
Vladimiro Sassone	vs@ecs.soton.ac.uk
Thomas Schwentick	thomas.schwentick@udo.edu
Paul Spirakis	p.spirakis@liverpool.ac.uk
Jukka Suomela	jukka.suomela@aalto.fi
Thomas Wilke	thomas.wilke@email.uni-kiel.de
Peter Widmayer	widmayer@inf.ethz.ch
Gerhard Wöeginger	g.j.woeginger@math.utwente.nl

Bulletin Editor: Kazuo Iwama, Kyoto, Japan
Cartoons: DADARA, Amsterdam, The Netherlands

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

All contributions are to be sent electronically to

`bulletin@eatcs.org`

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

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

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

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

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

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

The EATCS home page is `http://www.eatcs.org`

Table of Contents

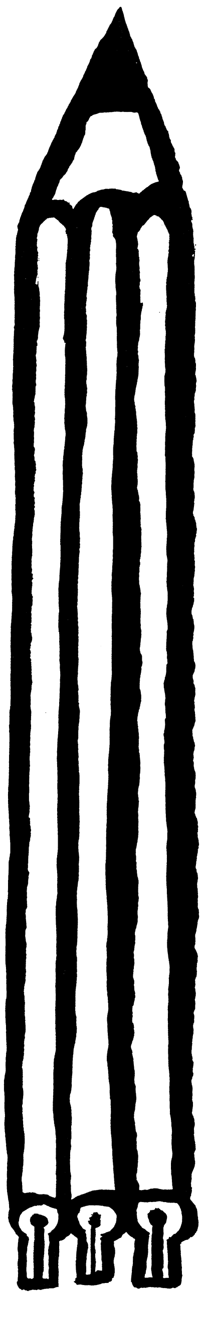
EATCS Matters

Dear colleagues,

I usually enjoy writing in all forms and I might even be considered a compulsive writer. However, for several reasons, this time I am finding it difficult to do so.

On a personal note, this is my last letter to you as president of the EATCS. At ICALP 2016 I will have served two terms at the helm of our association and I have decided not to run for a third term. I have thoroughly enjoyed working for the theoretical computer science community as a member of the EATCS Council for over ten years, and I have been truly honoured to serve as the president of association since July 2012, and to cooperate with the members of the council and with Ioannis Chatzigiannakis and Efi Chita at the EATCS Secretary Office. I have also had a very enjoyable and fruitful collaboration with Kazuo Iwama, the editor in chief of the Bulletin, under whose energetic leadership our flagship publication has consistently been of excellent quality. I have learned much from all of them.

However, the increasing pressures of our job are such that I feel that it is time for me to step down. I have probably given what I could to the EATCS, and our association will benefit from an influx of fresh blood and new ideas. The new leadership of the EATCS (president and vice-presidents) will be announced officially at the General Assembly at ICALP 2016 in Rome. I have no doubt that its members will make the EATCS more visible and influential than I have done. I will keep serving the EATCS as a loyal member

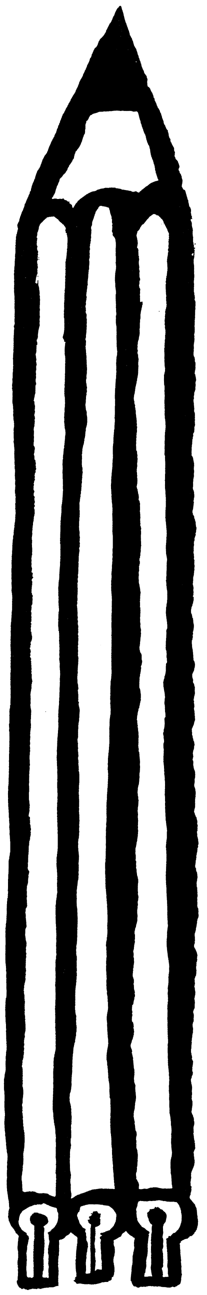


and an interested observer.

At ICALP 2016, we will also complete the process of changing the seat of the EATCS from Antwerp to Brussels. After a very long and sterling service, our treasurer Dirk Janssens will also take leave from the association. Dirk has been a stalwart of the EATCS and he deserves our most sincere thanks for all he has done over the years. Following his example won't be easy. Fortunately, our colleague Jean-Francois Raskin (Université Libre de Bruxelles) has agreed to take over the job of treasurer of the EATCS. On behalf of the EATCS, I thank Dirk and Jean-Francois, and wish them all the best for the future.

The recent, untimely passing away of several members of our community also makes the task of writing this letter difficult. This issue of the Bulletin of the EATCS features obituaries for Hartmut Ehrig, David Stifler Johnson and Helmut Veith. In addition, our good colleague and friend Zoltán Ésik passed away suddenly on Wednesday, 25 May, in the hotel room where he was staying with his wife during a visit to the research group led by my wife and me at Reykjavik University. He had delivered a survey talk at Reykjavik University on the "Equational Logic of Fixed Point Operations" on Tuesday, 24 May, and we were making plans for the coming days and for mutual visits over the next few months. Obituaries for Zoltán will appear in the October 2016 issue of the Bulletin.

We will remember these colleagues at ICALP 2016 and I encourage the members of the theoretical computer science community to honour their memory by studying their work and disseminating it amongst their



students. In 1918, the Italian poet Giuseppe Ungaretti wrote the following very short poem

Si sta come
d'autunno
sugli alberi
le foglie

which can be loosely translated as "We are like tree leaves in autumn". I learned that poem by heart like many other Italian school kids. It is only much later in life that one understands the true meaning of those words. However, the reaction of our community to the passing away of the above-mentioned colleagues has made me realize that the tree that hosted their leaves has very deep roots and that we can preserve those leaves for posterity.

Let's go back to core EATCS business. At this time of the year, the theoretical computer science community is about to meet at ICALP, the flagship conference of our association. As you know, the 43rd ICALP will take place in the period 12-15 July 2016 in Rome, Italy.
ICALP 2016 had the largest number of submissions in history (515 papers). Moreover, and most importantly, all tracks received many submission of very high quality. The PCs for the three tracks, which were expertly led by Yuval Rabani (Track A), Davide Sangiorgi (Track B) and Michael Mitzenmacher (Track C), did an amazing job in selecting an exciting conference programme.
I am very grateful to Tiziana Calamoneri, Irene Finocchi, Nicola Galesi, Daniele Gorla and their team for the extraordinary work they have done in organizing ICALP 2016.

The programme of ICALP 2016 will highlight research across many areas within theoretical computer science. I invite you to go to talks even outside your own research field. In particular, I hope that all participants will make a point of attending all the invited talks, which will be delivered by Subhash Khot, Marta Z. Kwiatkowska, Xavier Leroy and Devavrat Shah.

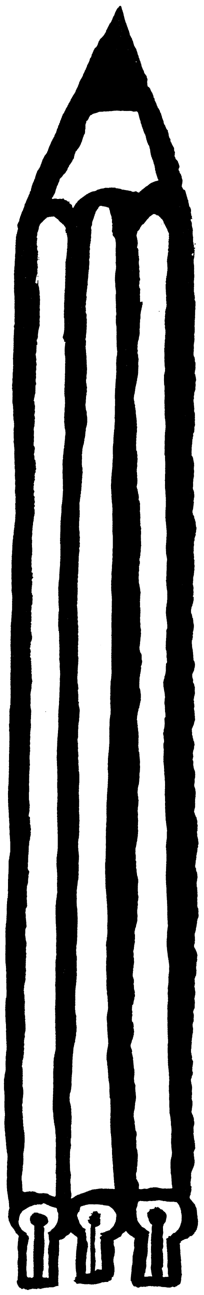
The best paper awards at ICALP 2016 will go to the following articles: Amplifiers for the Moran Process *by Andreas Galanis, Andreas Göbel, Leslie Ann Goldberg, John Lapinskas and David Richerby (Track A),* An Almost Cubic Lower Bound for Depth Three Arithmetic Circuits *by Neeraj Kayal, Chandan Saha and SÃľba stien Tavenas (Track A), and* Polynomial Time Corresponds to Solutions of Polynomial Ordinary Differential Equations of Polynomial Length *by Olivier Bournez, Daniel Graca and Amaury Pouly (Track B).*

The following papers will receive the best student paper awards: Analysing Survey Propagation Guided Decimation on Random Formulas *by Samuel Hetterich (Track A), and* An Optimal Dual Fault Tolerant Reachability Oracle *by Keerti Choudhary (Track C).*

Congratulations to the authors of the award-receiving papers!

As usual, a report on the conference will be published in the October 2016 issue of the Bulletin. I also hope that there will be some coverage of the event on the blogs devoted to theoretical computer science. I will issue a call for guest bloggers on my professional blog myself.

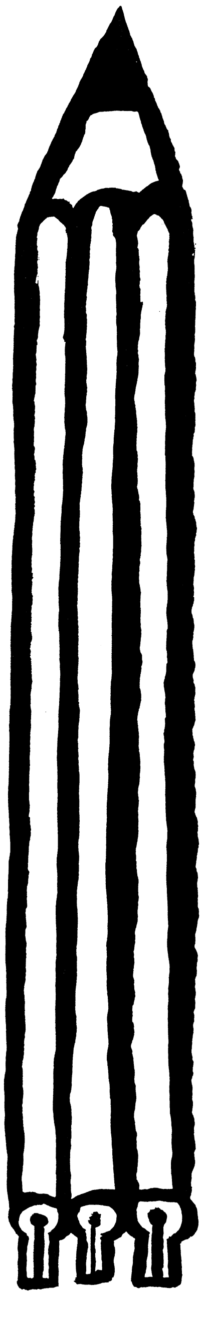
ICALP 2016 will be the first edition of the conference whose proceedings will be



available in open-access form in the LIPIcs series. The EATCS Council decided to move to open-access proceedings for its flagship conference as a service to the scientific community as a whole. I sincerely hope that readers of this letter will support this move, as well as our other activities, by becoming a member of the EATCS. This will allow us to use some of the financial resources of the association to cover the article processing charges for LIPIcs articles. Those charges will be increasing in the coming years, but will stay small and will guarantee free access to the archival version of the articles in the ICALP proceedings to researchers all over the world.

Apart from the invited and contributed talks, ICALP 2016 will feature the presentation of the EATCS Award 2016 to Dexter Kozen, of the Gödel Prize 2016 to Stephen Brookes and Peter O'Hearn, and of the Presburger Award 2016 to Mark Braverman. Moreover, during the conference, we will honour the EATCS Fellows class of 2016, who are -the late Zoltán Ésik (University of Szeged, Hungary) for "contributions to the fields of automata and formal languages, iteration theories, algebra and logic in computer science, and in particular to their connections. He has been able to apply deep theorems of some area to problems of other fields, yielding particularly short, beautiful and mathematically concise proofs."

-David Harel (Weizmann Institute of Science, Israel) for "fundamental contributions to program verification, database theory, and software engineering,



as well as for exceptional merits as a writer and teacher. The Statecharts model has had profound impact on software and systems engineering."

-Giuseppe F. Italiano (University of Rome Tor Vergata, Italy) for "fundamental contributions to the design and analysis of algorithms for solving theoretical and applied problems in graphs and massive data sets, and for his role in establishing the field of algorithm engineering."

-Kurt Mehlhorn (Max-Planck-Institut für Informatik, Germany) for "his influential contribution to the whole field of algorithmics over the past decades. In addition to key theoretical contributions, he has brought basic research closer to practice."

-Scott A. Smolka (Stony Brook University, USA) for "fundamental contributions to process algebra, model checking, probabilistic processes, runtime verification, and more recently for the successful application of most of these theories to cardiac-cell modelling and analysis."

Last, but not least, the EATCS Distinguished Dissertation Award Committee 2016 has selected the following three theses for the EATCS Distinguished Dissertation Award for 2016:

-Radu Curticapean, "The Simple, Little and Slow Things Count: On Parameterized Counting Complexity", -Heng Guo, "Complexity Classification of Exact and Approximate Counting Problems",

-Georg Zetzsche, "Monoids as storage mechanisms".

On behalf of the EATCS, I heartily thank the members of the award, dissertation and fellow committees for their work in the selection of this stellar set of award recipients and fellows. It will be a great honour to celebrate the work of these colleagues during ICALP 2016.

Even though ICALP 2016 has not taken place yet, we are already busy preparing future editions of the conference. As you may know already, ICALP 2017 will be held in Warsaw, Poland, in the period 10-14 July 2017. The conference chairs will be Mikolaj Bojanczyk and Piotr Sankowski. The PC chairs for the conference will be Piotr Indyk (MIT, USA) for Track A, Anca Muscholl (LaBRI and Université Bordeaux, France) for Track B and Fabian Kuhn (University of Freiburg, Germany) for Track C. The preliminary call for papers for the conference will be available in Rome. Moreover, during the general assembly at ICALP 2016, we will examine a bid to host ICALP 2018 in Prague, Czech Republic.

The EATCS continues to be active on increasingly many fronts. We have recently stipulated a new reciprocity agreement with the ACM Special Interest Group on Electronic Commerce (ACM SIGecom). We have also announced that the 2016 Alonzo Church Award for Outstanding Contributions to Logic and Computation will be given to Rajeev Alur and David Dill for their invention of timed automata. The Alonzo Church Award is a new major prize in cooperation with ACM Special Interest Group on Logic and Computation, the European Association for Computer Science Logic and the Kurt Gödel Society.

I interviewed the recipients of the 2016

Gödel Prize and of the 2016 Alonzo Church Award for the Bulletin of the EATCS. You can read the interviews in this issue of the Bulletin and I hope that you'll enjoy them as much as I did. I feel that they shed light on the development of the work that led to those awards and offer many general lessons about research from which we can all learn something new.

As usual, let me remind you that, at least until the middle of July this year, you are most welcome to send me your comments, criticisms and suggestions for improving the impact of the EATCS on the theoretical-computer-science community at president@eatcs.org.

I look forward to seeing many of you in Rome for ICALP 2016 and to discussing ways of improving the impact of the EATCS within the theoretical-computer-science community at the general assembly.

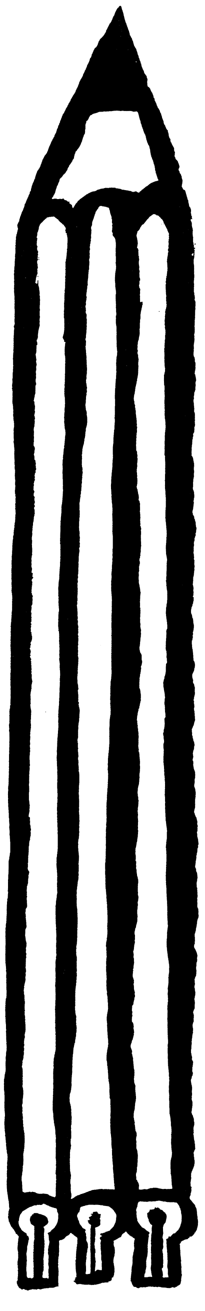
Luca Aceto, Reykjavik, Iceland
June 2016

Dear Reader,

Kyoto (and many other institutes in Japan) has a forced retirement system by age and it came to me this March. Fortunately I got a small position at RIMS in the same Kyoto University and am now spending almost the same daily life as before (well, virtually no classes or faculty meetings,... a bit lacking tensions). So I had a lot of things to do in February and March, mostly paper works.

Among others, I had to prepare a bunch of documents for my pension, which does not come to me automatically. Obviously it was a good chance to know what our pension system is like and especially what amount of money would be coming. Then it turned out that the description of those things is really complicated with several formulas. It is a well-known fact that only a very small group of people, mainly bureaucrats, do understand it and they do not want other people to do so. The reason is clear; they are afraid that many complaints are coming once people understand it.

However, after a couple of weeks' study, I finally ended up with the fact that our pension has a couple of different sources but the amount of money coming from its main part, which is also the hardest part to understand, is somewhat proportional to the income he/she has gotten in the whole carrier. This is surprisingly simple but the description never mentions that. Of course you can figure out the exact amount after understanding several definitions of terms and complex formulas, but its (pretty accurate) approximation is simple and

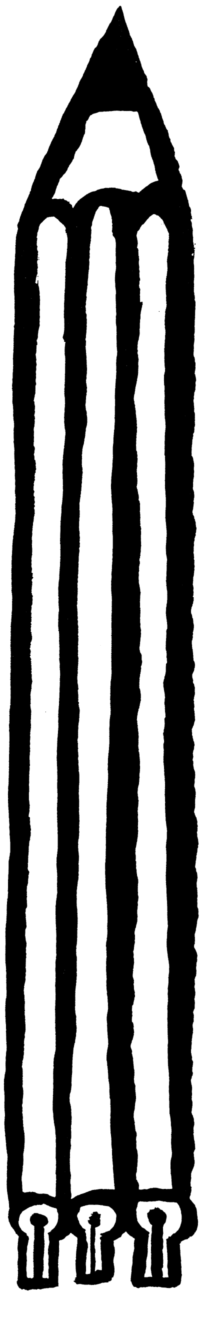


natural. But they still don't want to make it clear.

Then I noticed that exactly the same thing can happen when we write papers. I am sure there are many (but not all) cases that the basic idea is quite simple although the exact description should be complicated. Now there are several different types of authors. Some of them make it (the idea) clear but still succeed in claiming the result is nontrivial. Some of them deliberately hide it because they are afraid that its revealing would destroy the importance of the result. Some of them even do not know there is a simple idea behind the (rather complicated) result, which is sometimes indicated by reviewers. Our pension system is exactly the second type.

This issue of BEATCS has five obituaries including the one for David Johnson. I was a main organizer of SODA 2012 in Kyoto and I remember a lot of communications with David who everyone knows is a single main figure of the conference. He looked ok at that time, but soon after that I heard he was having a serious health problem.

I enjoyed reading advises from EATCS fellows. I also read similar articles here and there and even have written ones by myself. In such occasions, it is always hard, at least as far as I am concerned, to understand how the whole environments are different between now and when I was young (usually these kind of articles target young people). I can write how I should have done when I was young but I am not sure if this is useful at all for young guys of the current time.



Summer is approaching. Its image is different from place to place, but to me who lives in Kyoto it is definitely not very welcome, just heat and humid. I hope it IS welcome with you.

Kazuo Iwama, Kyoto
June 2016

Hartmut Ehrig (1944–2016)

Reiko Heckel (Leicester, UK)
with contributions by Andrea Corradini, Ugo Montanari,
Hans-Jörg Kreowski, Fernando Orejas and Grzegorz Rozenberg

Hartmut Ehrig passed away on March 17, 2016 at the age of 71. His colleagues and friends are mourning the loss of a most creative scientist and leader who made pioneering contributions to areas of theoretical computer science such as Categorical Automata Theory, Graph Transformations, and Algebraic Specifications, inspired generations of researchers and helped to build lasting communities.

Harmut was born in Angermünde (Germany) in 1944 and spent his academic career at the Technische Universität Berlin, where he studied Mathematics, Physics and Theoretical Informatics 1963 – 1969, worked as research assistant at the Mathematics Department 1970 – 1972, and received his PhD in 1971. In 1972 he was appointed Assistant Professor at the Informatics Department and received his Habilitation two years later. In that same year he was appointed Associate Professor of Theoretical Informatics. Hartmut became full professor at the TU Berlin in 1985 and held this position until he retired in 2010.

Early in his scientific career Hartmut developed a Unifying Theory of Automata, a categorical approach preceding computational models popular today based on monoidal categories and coalgebras. The categorical approach to semantics was a consistent theme throughout his work. In 1973 he applied it to the problem of formalising the notion of graph transformation, extending the definition of rewriting from formal (string) grammars to graphs. This led to the famous double-pushout (DPO) approach, which gave us one of the most recognisable diagrams in the application of category theory to computer science.

$$\begin{tikzcd}
L \arrow[d, "m"'] \arrow[dr, phantom, "(PO)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] \arrow[dr, phantom, "(PO)"] & R \arrow[d, "m^*"] \\
G & D \arrow[l, "l^*"] \arrow[r, "r^*"'] & H
\end{tikzcd}$$

This innovation, published in the 1973 paper Graph-Grammars: An Algebraic Approach with Michael Pfender and Hans-Jürgen Schneider, was instrumental in creating an entire discipline, variously referred to as graph grammars, graph rewriting or graph transformation, with a series of workshops and conferences that he helped to initiate, including the International Workshops on Graph Grammars and their Application to Computer Science with Volker Claus and Grzegorz Rozenberg starting in 1978 and the International Conferences on Graph Transformation (ICGT) since 2002. He was the first chair of the ICGT steering committee from 2000 to 2008.

Hartmut not only created and led the graph transformation community but made significant contributions to funding its operation through two European projects on Computing by Graph Transformation I and II. His monographs on Fundamentals of Algebraic Graph Transformation and most recently on Graph and Model Transformation as well as the Handbooks of Graph Grammars and Computing by Graph Transformation he helped create and edit remain part the core literature of the discipline.

Another area where Hartmut applied his style of categorical semantics are Algebraic Specifications. He developed the first compositional semantics for parameterised algebraic specifications based on amalgamation, a construction using a pushout in the category of generalised algebras (over arbitrary signatures). His monographs Fundamentals of Algebraic Specification 1 and 2, and Algebraic Specification Techniques and Tools for Software Development: The Act Approach became standard references and were widely adopted for teaching.

Hartmut started the series of international conferences on Theory and Practice of Software Development (TAPSOFT) in 1985 and helped its transformation into

ETAPS from 1998 onwards, today one of the most successful Computer Science conferences in Europe. Hartmut served as vice president of the European Association of Theoretical Computer Science (EATCS) from 1997 to 2002 and was vice president of the European Association of Software Science and Technology (EASST) since 2000.

Hartmut was amazingly productive. In addition to the editing and co-editing of more than 20 proceedings and handbooks, he authored and co-authored eight books and more than 400 papers in journals, proceedings, handbooks and other collective volumes, cooperating with more than 160 coauthors.

Hartmut supervised well over 50 PhD students, many now in senior positions themselves. When I came to Berlin in 1991 for my 3rd year of studies I was attracted by his genuine passion for science, which showed of course in Hartmut's own teaching but had also infected his entire group. After following a course on algebraic specifications and another one on graph grammars I was hooked and spent most of the rest of my studies there until a position in a research project was available and I became a PhD student under Hartmut's supervision. The following years were among the most intellectually inspiring and formative of my life. I not only learned from Hartmut the tools of the trade but also his approach to supervision. Hartmut thoroughly enjoyed working with students, hardly ever stopped discussing science, including over lunch and in the pub, and so gave us the impression that our work was, at least for now, the most important in the world. I have especially fond memories of Summer evenings in the Schleusenkrug (a pub by the canal locks in the Tiergarten), the group often including visitors, drawing diagrams on napkins over beer and Currywurst.

Hartmut's collegiality, integrity and genuine interest created a legacy, not only of his scientific work but also of his personal example and approach to research. It will remain with us, who were lucky enough to know and work with him.

Hartmut Ehrig

Ugo Montanari and Andrea Corradini
University of Pisa, Italy

As friends, collaborators and coauthors of Hartmut Ehrig we were heavily shocked by his departure.

One of the areas of interest for our scientific work is graph transformation. Hartmut has been the main originator and architect of this area. The idea was to extend to general structures, collectively called graphs, the constructions and the results already known for string, term and multiset rewriting. The deep knowledge and intuition Hartmut had in category theory guided him in developing, in 1973, together with Michael Pfender and Hans-Jürgen Schneider, the Double Push Out (DPO) construction. It works conveniently in several categories besides Graph and from the very beginning Hartmut started, with collaborators, the quest for the most general category where the interesting properties of DPO do hold, a quest eventually fulfilled recently by adhesive categories. To define a good notion of graph grammar was an open problem at the time. Ugo Montanari also contributed to it at some extent, together with John Pfaltz and Azriel Rosenfeld. When the DPO paper appeared, Azriel, a mathematician and a pioneer in picture processing and recognition, told Ugo he was definitely impressed by the clever and general construction of DPO, which was disclosing a new area of research.

Another milestone Hartmut contributed to the theory of graph transformation together with Hans-Jörg Kreowski was on concurrent rewriting. In 1978, when concurrency theory was in its infancy even for Petri nets, the notion of shift equivalence was a breakthrough. The extension of Petri Net theory (on multiset, or marking, rewriting) to graph transformation was developed later, mainly by Paolo Baldan, again with important contributions by Hartmut.

Another important result worth mentioning is on borrowed contexts, in collaboration with Barbara König. For a long time the DPO construction has been missing a notion of observation associated to a transformation. Observations are essential for defining abstract compositional semantics of processes. Following an approach by James Leifer and Robin Milner, Hartmut and Barbara introduced a categorical construction able to associate to a graph transformation an observation defined in terms of the minimal missing part necessary to apply a DPO.

Direct scientific work was by no means the only contribution Hartmut offered to our community. He was the main actor in coordinating the conference and workshop activity on graph transformation and in promoting EU working groups and projects in the area: COMPUGRAPH I (1989-1992), COMPUGRAPH II (1992-1996), GETGRATS (1997-2000) and APPLIGRAPH (1997-2000). Support for shared activities did allow for several visiting periods between partners. In particular, the connection between Berlin and Pisa was especially active and productive: in the period 1990-2000 three researchers from Pisa visited Berlin and three vice versa, for periods of about a year each on the average.

Among Hartmut's merits for the general computer science community, we want to mention explicitly the effort spent by Hartmut to organize the first TAPSOFT conference in Berlin, 1985. He recognised the need to connect theory with practice in the area of software development. Together with Maurice Nivat, the founder of EATCS and ICALP, Hartmut designed an articulated, flexible conference structure which turned out quite successful. The second TAPSOFT conference was in Pisa, 1987. Later, TAPSOFT evolved into ETAPS, presently one of the largest European conference addressing this mix of theory and practice.

We want to conclude these few words by remembering our frequent visits to

Berlin while collaborating with Hartmut and his group, for shorter and longer stays. Hartmut has always made us feel at home and, a proud host, was eager for us to enjoy the city, with all its contradictions.

Now most former students of Hartmut have left Berlin and have continued his work contributing to the development of several research centers in Germany and abroad. We will never forget the exceptional personal and scientific heritage of Hartmut Ehrig.

My 'giant' friend Hartmut

Hans-Jörg Kreowski
University of Bremen, Germany

Hartmut's death is very sad news for me as for many of his colleagues and friends. The graph transformation community and the algebraic specification community lost one of their pioneers, a leading, most inspiring and creative researcher, and guiding spirit to many of us.

As a student back in 1971, I attended Hartmut's seminar on categorical automata theory. Soon afterwards he introduced me to the fascinating world of graph transformation and supervised my PhD thesis. This was the beginning of a long, intense and fruitful period of cooperation and friendship in which we sat together for hundreds of hours discussing and working on categorical automata theory, graph transformation and algebraic specification. I owe a lot to him.

Hartmut spent all his academic career at the Technische Universität Berlin only interrupted by longer research stays at Amherst, Yorktown Heights, Los Angeles, Leiden, Barcelona, Rome and Pisa. Besides teaching and research, he was also deeply involved in university affairs serving repeatedly as department chair and leading the Institute for Software Engineering and Theoretical Computer Science for 32 years. I remember well the long hours in the 1970s sitting together with Hartmut and discussing the pressing issues of departmental politics (and there were many thrilling and conflict-laden topics to discuss at that time).

Hartmut was a most productive editor and co-editor, author and co-author. It was far from easy to keep pace with him. Not only the sheer amount of printed outcome is striking, but also the fact that he was the driving force behind most of his publications. If he looked into a matter, then he did not stop before he understood it in depth. In this process, he often came up with innovative formulations, views and approaches. He was a profound thinker who worked hard to disseminate his ideas. He was a great communicator attending a good many conferences, visiting numerous research groups all over the world and inviting a great number of famous and promising scientists to Berlin.

Hartmut was also a dedicated teacher who prepared many courses in Theoretical Computer Science and Mathematics for Computer Scientists including teaching materials as evidenced by his text book *Mathematisch-strukturelle Grundlagen der Informatik* and a wealth of unpublished lecture notes. At his university, he belonged to the minority of professors who experimented regularly with new principles of teaching. He invested much time and effort in the supervision of his students. In particular his over 50 PhD students always found him ready to advise and collaborate.

Hartmut largely personifies the area of graph transformation. But appreciating his achievements and the services rendered to graph transformation, one should not forget that he played a similar role in algebraic specification and contributed significantly also to the areas of automata theory, Petri net theory and formal and visual modeling. Beeing one of the most influential scientists in Computer Science for some decades, he was a nice, friendly, generous, reliable, and faithful colleague and friend. I have always admired his can-do attitude.

There is a very old metaphor going back to the twelfth century that we can see further making progress in science because we can stand on the shoulders of giants. Hartmut was, is and will be such a giant for our scientific community. What better way to honour him and his work than to follow his footsteps and aspire to achieve his level of service and dedication. I mourn for my 'giant' friend.

Hartmut in Barcelona

Fernando Orejas
UPC, Barcelona, Spain

In July 1988 Hartmut visited Barcelona for three weeks, because he was interested in working with us on our approach to behavioral algebraic specifications, developed by Pilar Nivela in her thesis. I had met Hartmut six years before at ICALP 82 in Aarhus, but I knew his work from some years before. My view of algebraic specification was very close to his. Actually, I had done some work on the composition of implementations, based on his notion of algebraic implementation. This visit was part of his summer vacation. While we were working in Barcelona, his family was somewhere in the Costa Brava (at some point, he joined his family for a few days and then returned to Barcelona).

I remember that, for me, this visit was a kind of nightmare, but a very nice nightmare that we both enjoyed very much. From the scientific viewpoint we did some interesting work, but we conjectured a main result that we found was false by the end of his stay. In addition, during his visit, work was very stressing for me. At the time, I had some other work commitments that I had to fulfill in parallel, while he was on vacation with no other duties. As a consequence, our day scheduling was roughly as follows. I was waking up early (around 7 AM) and working in my other duties until 10. At that time he was showing up in my office, and we were working until 1 PM, when we were going for lunch. After lunch, he was going to the residence where he was staying to sleep a siesta. Meanwhile, I was working on my other duties, until he would show up again. Then, we were working until 7 or 8 PM, when we were going for dinner to some nice restaurant and, afterwards, to some nice bar for a drink, which meant going to sleep typically later than 2 AM. I remember that one night (fortunately it was Friday, so we were not working the following day), we were a group of people having an excellent time in a beautiful open air bar, and it was around 5 AM. Since I was quite tired I decided to suggest to leave and to go to sleep. But instead of saying it explicitly, which is a bit rude, I did it in a polite way. What I said was something like *Our glasses are empty so, either we leave, or we ask for new drinks*. His answer was *Oh! I'm enjoying this place*. So, we asked for new drinks and we left the bar around 6.

We also had some problems in the social activities of his visit. I especially remember his first day in Barcelona. We had made a reservation on a very nice seafood restaurant, but too late we discovered that he did not like seafood (or that's what he claimed) and that in this restaurant there were no meat dishes. At the end, he said that he would have some salad as first, dish and paella as main dish, although he also claimed that he did not like paella, because years before he had a terrible paella in some restaurant (probably a tourist trap). As for the rest of us, we decided to have some very nice tapas to share, first dish and then a couple of different rice dishes as mains. When all these tapas arrived to the table, he looked at them and to his boring salad and offered to us to share his salad, if he was allowed to share our tapas. At the end, he had enjoyed the seafood and the paella. Some time later he told me that he still did not like seafood, unless I would assure him that it was good. Actually, years later, in another visit, he even had a black paella, because I told him that he would like it, which he did.

Despite these initial "problems", since we both enjoyed working together, we started a long and fruitful collaboration and a very good friendship. Every two years, more or less, he was coming to Barcelona and in the complementary years I was visiting Berlin. According to DBLP, we coauthored 58 papers. Actually, he is my main coauthor and I am his main coauthor, with respect to the number of papers coauthored. Initially, we started working in algebraic specification, but later he introduced me to graph and model transformation and I introduced him to logic programming and some logical techniques for automated deduction. Anyhow, I think that I learned much more from him than he learned from me.

In the last three or four years, regrettably, our collaboration ended when Hartmut was unable to come to Barcelona because of problems with his mobility and new duties at my university made it difficult to make time for a stay in Berlin. Nevertheless, I was planning a visit when I learned the sad news that I had lost a very good friend.

Remembering Hartmut

Grzegorz Rozenberg
Leiden University, The Netherlands
University of Colorado at Boulder, USA

We were close (also family) friends for over 40 years. We did joint research, cooperated on editing books (proceedings, handbook, ...), and on establishing a forum for researchers on graph transformation (first a workshop which then evolved into a conference). I also worked closely with Hartmut on producing some of his books.

We visited each other quite often (especially in the 1970s through the 1990s). I still remember staying in his apartment in the 1970s from where I had a bird's eye view of the Berlin wall.

Hartmut was a passionate scientist and a 100% reliable partner. Whatever project we were working on he would be a real motor behind it - it was simply not possible to stay behind when working with Hartmut. Also, whatever part of a common project he took over, I knew in advance that it would be implemented ahead of the agreed deadline.

I formed important "friendship links" through Hartmut. Once he invited me to Berlin with an explicit request to work with his very talented Ph.D. student Hans-Jörg Kreowski. This stay became the beginning of my long scientific collaboration and personal (family) friendship with Hans-Jörg.

Hartmut enjoyed very much my magic shows, especially the illusions involving unexpected/amazing coincidences. The last unexpected coincidence took place on March 17, 2016 when I wrote an email to Hartmut about his last book. This was the day that he passed away (nobody except for his close family knew about his illness).

I lost a dear friend. Scientific communities thrive because they include individuals such as Hartmut. We are all indebted to him and he will be warmly remembered by many.

HELMUT VEITH (1971–2016)

Thomas Eiter
TU Wien
thomas.eiter@tuwien.ac.at

Richard Zach
University of Calgary
rzach@ucalgary.ca

TU Wien and the Faculty of Informatics mourn the loss of Prof. Helmut Veith, who passed away on March 12, 2016 at the age of 45. He fell into a coma due to unforeseen complications following routine surgery; he died without regaining consciousness.

Education and Career

Helmut Veith was born on February 5, 1971. After graduating from high school at the BG/BRG Tulln in 1989, Veith studied computational logic at the TU Wien and graduated in 1994. He received his doctorate in computer science in 1998, the promotion ceremony was carried out "sub auspiciis praesidentis" by the president of Austria, a rare honor indicating his immaculate academic record. Helmut Veith began his scientific career at the Institute of Information Systems, where he held a position as assistant researcher (Universitätsassistent) in the Databases and Expert Systems Group from 1995 onward. In 2001, he received the Habilitation, the right to teach at the university level, for the field of applied and theoretical computer science. Shortly thereafter, Veith accepted a position as associate professor (Professor C3) at the Technical University of Munich in 2003, which he held until 2007. From 2008 to 2009, he began building a large research group at the Technical University of Darmstadt, were he was full professor (Professor W3). He returned to the TU Wien in 2010, and was the inaugural holder of the professorial chair in computer aided verification.

Helmut Veith became interested in computer aided verification in the late 1990s after taking a guest lecture course in model checking given by Orna Grumberg. His interests expanded during a post-doctoral visit at Carnegie Mellon University in 1999/2000, supervised by Turing Award-winner Edmund M. Clarke, and funded by a Max Kade Fellowship. Prof. Veith's subsequent work focussed on this area. He established a sustained and successful research collaboration with Prof. Clarke, and was named Adjunct Professor at Carnegie Mellon in 2005.

Awards and Memberships

Helmut Veith was a leading researcher in the area of computer-aided verification, to which he made numerous contributions. He worked in particular on model checking for software and hardware, on abstraction tools, parametric questions in model checking, analysis and testing of computer programs, and in this context also on temporal logic. Two examples can serve as illustrations. The first is a path-breaking, outstanding article on the verification of systems using refined abstraction; it is universally known in the field as the "counterexample-guided abstraction and refinement (CEGAR)" method and has found applications in many areas outside as well. This work received the CAV Award 2015, an award that honours contributions of fundamental importance to the field of computer aided verification. The second is a paper on the verification of modular software, which received the ACM SIGSOFT Distinguished Paper Award in 2004.

Helmut Veith's research interests, however, were not limited to verification. His wider interests included computer security and embedded systems, mathematical logic (especially fuzzy logics), the theory of databases, finite model theory, and complexity theory. He made significant contributions to each of these fields. He published his results in numerous papers in the most prestigious journals and conference proceedings. Due to his interdisciplinary approach to research, driven by curiosity and applications, Helmut Veith was well-known and admired in several research communities. This is reflected in his membership in many program and steering committees. In his main field of research, Helmut Veith was co-editor of the Handbook of Model Checking, the last volume of which is scheduled to appear this year, and co-chair of the CAV conference program committee in 2013 as well as of the FMCAD conference this year.

Contributions to the TU Wien and Science in Austria

Following his return to the TU wien, Helmut Veith considered it his challenge to integrate and foster the significant existing potential in areas related to logic, such as databases, artificial intelligence, knowledge-based systems, automated deduction, with computer-aided verification. His aim was to bring these areas together under the banner of logic in computer science, and to make Vienna and Austria an international center for logic and verification. In this regard, his efforts to create an Austrian research network on "Rigorous Systems Engineering (RiSE)" have to be highlighted; the network has been funded by the Austrian Science Fund since 2011, and he served as its vice chair. At the TU Wien he co-initiated a doctoral college in logic in computer science. After its successful completion, it transformed into a doctoral program in "logical methods in computer science", also funded by the Austrian Science Fund beginning in 2014; Veith served as its director. This doctoral program made a substantial contribution to a restructuring of graduate training in the Faculty of Informatics. Other projects suggested or initiated by Veith have been the Vienna Center of Logic and Algorithms (VCLA), a platform for fostering national and international research and collaboration in the area of logic and algorithms, and the Vienna Summer of Logic, the largest event in the history of logic. This unprecedented conference united in one place a large number of the most important annual meetings in the areas of mathematical logic, logic in computer science, and logic in artificial intelligence, which enabled a broad exchange of research results. It was held in July 2014 at the TU Wien. The event, which was shaped in large part by Veith as its co-chair, was an enormous success and was received enthusiastically by the research community.

Helmut Veith not only excelled in his research and organisational projects. He was unfailing also in his dedication to the Faculty and its interests. He held a central role in the development and organization of the Faculty, where he participated in numerous working groups and committees. His inexhaustible creativity and his imagination showed new avenues. He proved this already as an undergraduate student through his development of the studium irregulare in "computational logic".

Helmut Veith was an outstanding proponent of foundational research, but he also understood the importance of innovation and application. He was always open to cooperation with other disciplines. Currently an interdisciplinary project with the Faculty of Architecture, which Veith helped bring about, and which is funded by the Austrian Science Fund through the PEEK program, serves as further evidence of the wide horizon of his interdisciplinary thinking. He was going to contribute his own expertise in information design to this project.

Nurturing New Talent

Teaching, learning, and nurturing new academic talent were a particular and important concern for Veith. He contributed to the design of masters and doctoral programs, and supervised numerous masters theses and dissertations, many of which received awards. He also supervised a number of post-doctoral researchers and supported their promising careers. He is appreciated especially by the numerous young scientists who received research awards thanks in part to his dedicated mentorship. Helmut Veith was researcher and teacher with heart and soul. With his death, the Faculty of Informatics and the TU Wien has lost one of its most outstanding and innovative leaders. The Austrian science community, and indeed the international computer science community, has lost a highly respected and influential member. He was also a well-rounded academic with interests, e.g., in literature and performing arts.

Helmut Veith was a cooperative and open colleague and a very good friend to us all. His death leaves a void that will never be filled; we will miss him. It is incomprehensible that he would be taken from us in the prime of his life. He already created so many things, but there were also so many hopes. His work and legacy will be our mission.

The Faculty of Informatics at TU Wien will forever honor his memory. Our thoughts are with his family during this difficult time.

Helmut Veith (1971–2016)

Richard Zach
University of Calgary
rzach@ucalgary.ca

My friend and colleague Helmut Veith[1] died on Saturday, 12 March 2016. His death is a great and shocking loss to his family and friends, and the logic community, especially in Austria.

I've known Helmut since we were undergraduates in computer science at Vienna Technical University in the early 1990s. We shared a passion for theoretical topics in computer science, a love of Robert Musil; we took many courses together. In fact, we liked logic so much that together we created a specialized course of study (a *studium irregulare*) in computational logic. At the time this still required approval by the federal ministry of science and research, and it was a lot of work, but we got it approved. It has since morphed into a standard stream in the computer science curriculum at the TU Vienna, and more recently a doctoral program, all in no small part due to Helmut's tireless organizational work. I was a year ahead of him, but he was the better student. He literally had straight *A*s throughout high school and university. In Austria, that earns you a doctorate *sub auspiciis praesidentis*, and the president of the republic himself hands you your diploma. His *Diplom* was on finite model theory; his dissertation on the complexity of database query languages (supervised by Georg Gottlob). Helmut had a stellar career: appointments at TU Munich, TU Darmstadt (two of the centers of computer science in Germany), and finally a full professorship at our alma mater in 2010; add to that an adjunct professorship at Carnegie Mellon. Not only was he the better student, he had the better sense to stay in computer science, and to do something useful with logic. He was one of the leading experts in computer aided verification, especially model checking, with over 120 papers to his name. After his return to Vienna, he was instrumental in getting the Vienna Center for Logic and Algorithms[2] off the ground, led the organization of the Vienna Summer of Logic[3], and helmed the Austrian doctoral program on logical methods in computer science[4]. Helmut wasn't just an outstanding researcher, he was also

[1]http://forsyte.at/people/veith/
[2]http://www.vcla.at/
[3]http://www.vsl2014.at/
[4]http://logic-cs.at/phd/

passionate about improving undergraduate education in logic and computer science (he served on the ASL's Logic Education committee, and we co-organized a special session at the 2014 Logic Colloquium), about diversity in the field, and about science policy.

We need more people like him. I miss him.

For Helmut Veith (1971–2016) "I have this idea"

Oliver Lehmann
IST Austria
oliver.lehmann@ist.ac.at

My friend Helmut Veith passed away on Saturday, 12 March 2016, from the effects of a pulmonary embolism after an operation on his leg. He was 45 years old. Helmut is survived by his wife Anna, their son Nikita, and his mother Herta. Our thoughts are with them. I leave it to others to honour his pioneering achievements in the field of computer science. With this text I intend to recognize his outstanding abilities as science communicator.

It sounds trivial to state that Helmut Veith led a binary life. As a computer scientist Helmut would deal with decisions described by yes and no, plus and minus, 1 and 0. But If I say binary and mention Helmut I add the attribute of quality to this binary decision-making. Helmut was a man of a distinct "Möglichkeitssinn" in the sense of Robert Musil, an author—among many others—he loved as Richard Zach reminded us in his befitting obituary[1].

Options, chances, possibilities do not describe properly what fascinated Helmut when faced with a problem. It was the potential the problem offered and which he detected with his "Möglichkeitssinn" which Musil explains as follows: "A possible experience or truth is not the same as an actual experience or truth minus its reality value, but has—according to its partisans, at least—something quite divine about it, a fire, a soaring, a readiness to build and a conscious utopianism that does not shrink from reality but sees it as a project, something yet to be invented"[2]. For Helmut, the answer to a riddle was wasted (or at least bland) if the answer did not enable new paths of thinking, new fields of research, new ways of communication. A solution was not the simple confirmation of feasibility but a proposal for improvement.

His peers have described Helmut's merits as a computer scientist[3] in an ap-

[1]Note of the editor: Richard Zach's obituary is also published in this issue and is available as a blog post at http://richardzach.org/2016/03/13/helmut-veith-1971-2016/.

[2]Musil, R. (1996) "The Man Without Qualities", translated by Sophie Williams and Burton Pike, Vintage Books, p. 11

[3]https://www.tuwien.ac.at/de/aktuelles/news_detail/article/10002/

propriate manner. I will focus on his interest in communicating his science to all kinds of public. His unique quality to strive for improvement contributed to Helmut's decision to wade into the sphere of communication—including the option of failure.

Let me illustrate this with our first collaboration: obviously, just putting out a press release on the establishment of a network bringing together all Austrian research groups involved in Rigorous System Engineering was not enough for him. It had to be more like convincing the Austrian public that Turing Award winner Ed Clarke who had travelled to Austria for the kick-off meeting of the new platform was headline stuff. For the press conference with Clarke in March 2011 not only the president of the Technical University of Vienna was convinced to participate but also the federal science minister. A trendy location with a good selection of food and wines was chosen and the time set in such manner that the journalists could use the press conference as their first watering hole after that day's deadline. The evening nevertheless ended in a disaster. In short: Ed Clarke and Rigorous System Engineering could not compete with Fukushima that had blown its fuses and the headlines twelve hours before.

The reason why this first joint attempt in communication did not fail was a media workshop a couple of months before the press conference. Helmut had first supported and then pushed (including supplying his own coffee machine for the coffee break, if I remember correctly) the, at least for Austria, new idea of this workshop. At this workshop in 2010 at the TU Wien he—and other scientists he had led by example—offered a carefully selected group of journalists the most precious commodity they had available: time and attention. The effort paid off. In spite of the Fukushima disaster, Rigorous System Engineering made it into the editorial offices and continues to do so.

It must have been this experience that science no matter how complex can be communicated if the right components were sensibly combined that drove him to continue with this venture: "Du, ich hab da eine Idee" ("Listen, I have this idea") would be his usual opening line, followed after the explanation by a "What do you think?" Like a bug, or rather like a spell, that enchanted him and inspired others he conveyed his vision usually already focused onto a target. The Vienna Summer of Logic 2014[4] started with something along these lines. Again, this event has been praised rightly by Helmut's peers for its sheer numbers and the fact that it brought the diversity of logic and its conferences into one event. From the many components let me just pick the most significant for success in terms of communicating science:

- the comprehension that in a visual world the choice of a logo and of key visuals is as important as any written content;

[4]`http://vsl2014.at/`

- the requirement of extensive PR activities based on a concise strategy;
- the necessity that if the public won't come to logic, logic must go public;
- the insight that with promoting a specific field of science we also communicate science as such.

Some hyperbolic ideas worked, like convincing the editors of Austria's leading current affairs publication *profil* not only to produce an extensive report on the "most important field of science in the world" (what else?) but actually to put 21 logicians working in Austria onto its cover. Some other hyperbolic ideas didn't, like convincing the local district assembly at very short notice to name a street after Kurt Gödel. By the way, as an act of acknowledgement of Helmut's achievements this should be rectified in due course.

Maybe the Logic Lounge[5] summarized Helmut's intentions in the best manner: a meeting place for friends of logic in the very public of a trendy club in the centre of Vienna offering "insights into the millennium old discipline of logic, celebrating the antique concept of the philosophical symposium (from the Greek word συμπίνειν (sympinein, 'to drink together'))" as we wrote on the website, hosted not by an expert but an interested person, most likely a journalist able to bridge the gap between the scholar and the public. The subjects ranged from the history of logic at Vienna universities in the 1980s presented by Georg Gottlob with surprise appearances of veterans like media artist Peter Weibel to Christos Papadimitriou on his graphic novel and NYT bestseller "Logicomix", Moshe Vardi on the ethics of AI, Richard Zach on Gödel$^{\text{TM}}$s incompleteness theorems, Byron Cook on the logic of symbols and, finally, Ruzica Piskac, Magdalena Ortiz and Irene Schreier Scott on female logic. It is fair to assume that science communication in Austria cannot go behind this conference and its events in terms of intellectual capacity, diversity, accessibility and cultural impact.

I am skipping numerous other ideas in different degrees of realization like rewriting first the textbooks and then the curriculum for high school students after examining the teaching material in maths with which his son had been supplied or the support of developing "Informatik Austria", again a network of research groups in computer science, for which Helmut generously sacrificed his time and creativity. And Helmut was equally generous when it came to support ideas of others. Of course, his Vienna Center of Logic and Algorithms participated in the Vienna Ball of Sciences[6] from the very first moment onwards, with Helmut explaining the impact of probability and game theory at the roulette table. The idea of the ball was a setting proper to his liking: think big, think better, and do not refrain from challenging common expectations or attitudes.

[5]`http://vsl2014.at/public-events/index.html`
[6]`http://www.wissenschaftsball.at/`

Finally, I wrote the first draft of this text sitting in the Café Wortner close to his office at the TU where Helmut and I had wanted to meet to discuss another idea: The Südbahn-Hotel had gone up for sale. Why not convince the federal ministry, the Austrian universities, and the provinces of Lower Austria, Styria and Vienna to acquire this inspiration for the "Grand Budapest Hotel" and turn it into a meeting, retreat, vacation and conference space for interdisciplinary research in central Europe? Heck, why not?

So what does Helmut leave us? The notions that it is worth

- to encourage scientists to communicate their findings—if not in all appropriate scientific detail then at least by the way of sharing their enthusiasm and their zeal for their subject;
- to persuade stakeholders in science management and politics to support basic research not just for the sake of material exploitation (that is fine too) but that doing science contributes immensely to the fabric of society and is a genuine part of our way of life;
- to convince the media to invest time and attention into complex subjects which do not give themselves away instantly to immediate comprehension but are worth some thorough research as the subjects open whole new spheres of understanding (plus headlines and covers);
- to motivate the public in the context of the societal concept of participation to become a part of an enlightening sphere where visions are welcome, problems can be handled, conflicting opinions merged for improvement of the status quo, and ideas ventured without fear.

Sounds all too pompous?

Just imagine Helmut saying: "Listen, I have this idea."

About the author Oliver Lehmann is Chair of the Austrian Association of Education and Science Journalists, Head of Stakeholder Relations at IST Austria, and was the Media and Public Affairs Chair of the Vienna Summer of Logic 2014.

David Stifler Johnson: A Tribute by Lance Fortnow

On the morning of March 9, 2016 I heard the terrible news from Zvi Galil of David Johnson's passing the previous day ending his struggle with cancer. I quickly wrote up a blog post that I reprint below. A decade ago I had plans to write a book on the history of the P versus NP problem and interviewed a number of theorists including David Johnson. I found my old notes from a phone conversation I had with David on August 2, 2005 about his early days. The notes aren't complete but I'll try and tell his story the best I can. I apologize in advance for any mistakes that follow.

David Stifler Johnson was born on December 9, 1945 in Washington, DC. He graduated with a bachelor's degree from Amherst College in 1967. He was interested in artificial intelligence and went to MIT to get a Master's degree. Theoretical computer science was just getting started back then and David was in the mathematics department thought he took courses in automata theory. A year and a half later he received his MS degree with a thesis on average case analysis of tree search algorithms.

While in Korea serving as an officer in the US Army, David read about Minsky and Papert's work on perceptrons and would return to MIT for a PhD in the fall of 1971. David talked about many of his fellow students at MIT: Terry Winograd was a fellow math student, Larry Stockmeyer, Nick Pippenger, Joel Seiferas, Nancy Lu. David, an avid runner even back then, had races inside around the corridors of MIT.

Albert Meyer and Patrick Fischer taught algorithms from the first two volumes of Knuth. From that class David Johnson learned about bin packing that led to his thesis on approaches to approximating optimal solutions for that problem. David Johnson also started thinking about online versus offline algorithms around that time.

David Johnson started his PhD studies shortly after Cook had his seminal paper on NP-completeness. David attended that IBM hosted Symposium on Computer Computations in March of 1972 where Karp presented his twenty-one NP-complete problems. David said that already at that point viewed NP-completeness as important but Karp "brought it home". Hopcroft during that meeting said, in what would be a massive understatement, that he didn't think P versus NP would be solved in ten years. There was a workshop at MIT's Endicott house on NP-complete problems that really got David interested in the area.

In 1973, David Johnson received his PhD from MIT and went to work at AT&T Bell Labs. He worked for AT&T in its various guises for forty years until taking on his final position at Columbia. He fondly remembers 1980-81, the year he took visiting the University of Wisconsin in Madison working with Larry Landweber and others.

Karp didn't use the term NP-hard and NP-complete in his paper. Donald Knuth ran a poll to come up with good terminology. Knuth discusses the results of the poll in an essay in the January 1974 SIGACT News and after describing a number of humorous proposals

> The "winning" write-in vote is the term NP-hard, which was put forward mainly by several people at Bell Labs, after what I understand was considerable discussion...This term is intended for use together with another new one, NP-complete, which abbreviates 'polynomial complete' and is the same time more exact.

So not only did David Johnson and Michael Garey go ahead and write their incredible book on NP-completeness, they actually helped name the field.

The rest is the text based on my blog entry
http://blog.computationalcomplexity.org/2016/03/david-johnson-1945-2016.html.

David Johnson, a leader and advocate for algorithms and all of theoretical computer science, passed away March 8, 2016 at the age of 70. A truly sad day for us all.

David's 1979 book with Michael Garey, Computers and Intractability: A Guide to the Theory of NP-Completeness, is still the best reference on the topic and perhaps the single most important resource in any computer scientist's library. David Johnson also wrote the NP-completeness column for the Journal on Algorithms and later the ACM Transactions on Algorithms, as well as "A Catalog of Complexity Classes" for the 1990 Handbook of Theoretical Computer Science. David founded the Symposium on Discrete Algorithms (SODA), a conference that is now often mentioned with STOC and FOCS as a top theory venue. He created the DIMACS algorithms challenges. He led SIGACT from 1987-1991, really transforming that organization, and served as its face for many years thereafter. I'm only scratching the surface of what he's done for the community, and can think of no one who put more effort into making the theoretical computer science as strong as it is.

Of course David was a great researchers as well, working on NP-completeness and approximation algorithms.

He received an ACM Fellow in 1995, the first SIGACT Distinguished Service prize in 1997 and the Knuth Prize in 2010. He used his Knuth prize lecture to push for practical applications for our algorithms. Just a month before he died he was elected into the American National Academy of Engineering.

I worked with David Johnson closely on various SIGACT activities. David never missed a STOC and we always invited him to the SIGACT Executive Committee dinners, not because he had an official role, but because he was David Johnson. I truly respected and admired David and glad I could call him a friend. We'll miss him deeply. STOC and SODA just won't be the same without him.

Institutional Sponsors

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

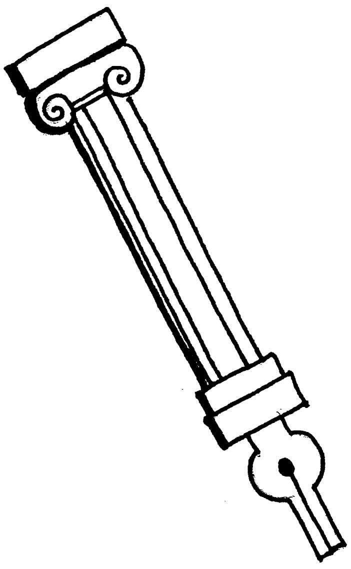
CWI, Centum Wiskunde & Informatica
Amsterdam, The Netherlands

MADALGO, Center for Massive Data Algorithmics
Aarhus, Denmark

Microsoft Research Cambridge
Cambridge, United Kingdom

Springer-Verlag
Heidelberg, Germany

EATCS Columns



The Distributed Computing Column

by

Stefan Schmid

Aalborg University, Denmark
schmiste@cs.aau.dk

Survey of Distributed Decision[*]

Laurent Feuilloley and Pierre Fraigniaud

Institut de Recherche en Informatique Fondamentale
CNRS and University Paris Diderot

Abstract

We survey the recent distributed computing literature on checking whether a given distributed system configuration satisfies a given boolean predicate, i.e., whether the configuration is legal or illegal w.r.t. that predicate. We consider classical distributed computing environments, including mostly synchronous fault-free network computing (LOCAL and CONGEST models), but also asynchronous crash-prone shared-memory computing (WAIT-FREE model), and mobile computing (FSYNC model).

1 Introduction

The objective of this note is to survey the recent achievements in the framework of *distributed decision*: the computing entities of a distributed system aim at checking whether the system is in a legal state with respect to some boolean predicate. For instance, in a network, the computing entities may be aiming at checking whether the network satisfies some given graph properties.

Recall that, in a *construction task*, processes have to collectively compute a valid global state of a distributed system, as a collection of individual states, like, e.g., providing each node of a network with a color so that to form a proper coloring of that network. Instead, in a *decision task*, processes have to collectively check whether a given global state of a distributed system is valid or not, like, e.g., checking whether a given coloring of the nodes of a network is proper [25]. In general, a typical application of distributed decision is checking the validity of outputs produced by the processes w.r.t. a construction task that they were supposed to solved. This applies to various settings, including randomized algorithms as well as algorithms subject to any kind of faults susceptible to corrupt the memory of the processes.

[*]Both authors received additional support from Inria project-team GANG.

The global verdict on the legality of the system state is obtained as an aggregate of individual opinions produced by all processes. Typically, each process opinion is a single bit (i.e., *accept* or *reject*) expressing whether the system state looks legal or illegal from the perspective of the process, and the global verdict is the *logical conjunction* of these bits. Note that this mechanisms reflects both decision procedures in which the individual opinions of the processes are collected by some centralized entity, and decision procedures where any process detecting some inconsistency in the system raises an alarm and/or launches a recovery procedure, in absence of any central entity. We will also briefly consider less common procedures where each process can send some limited information about its environment in the system, and a central authority gathers the information provided by the processes to forge its verdict about the legality of the whole system state.

The difficulty of distributed decision arises when the processes cannot obtain a global perspective of the system, which is typically the case if one insists on some form of locality in networks, or if the processes are asynchronous and subject to failures. In such frameworks, not all boolean predicates on distributed systems can be checked in a distributed manner, and one of the main issue of distributed decision is to characterize the predicates that can be distributedly checked, and at which cost. For predicates that cannot be checked, or for which checking is too costly, the system can be enhanced by providing processes with *certificates*, with the objective to help these processes for expressing their individual opinions. Such certificates could be produced by an external entity, but they might also well be produced by the processes themselves during a pre-computation phase. One typical framework in which the latter scenario finds application is self-stabilization. Indeed, a self-stabilizing algorithm may produce, together with its distributed output, a distributed certificate that this output is correct. Of course, the certificates are also corruptible, and thus not trustable. Hence, the checking procedure must involve a distributed verification algorithm in charge of verifying the collection of pairs (output, certificate) produced by all the processes. Some even more elaborated mechanisms for checking the legality of distributed system states are considered in the literature, and we survey such mechanisms as well.

We consider the most classical distributed computing models, including synchronous distributed network computing [49]. In this setting, processes are nodes of a graph representing a network. They all execute the same algorithm, they are fault-free, and they are provided with distinct identities in some ID-space (which can be bounded or not). All processes start simultaneously, and computation proceeds in synchronous rounds. At each round, every process exchanges messages with its neighboring processes in the network, and performs individual computation. The volume of communication each node can transmit and receive on each of its links at each round might be bounded or not. The CONGEST model typically assumes that at most $O(\log n)$ bits can be transferred along each link at each round in n-node networks. (In this case, the ID-space is supposed to be polynomially bounded as a function of the network size). Instead the LOCAL model does not limit the amount of information that can be transmitted along each link at each round. So, a t-round algorithm $\mathcal{A}$ in the LOCAL model can be transformed into another algorithm $\mathcal{B}$ in which every node first collects all data available in the ball of radius t around it, and, second, simulate $\mathcal{A}$ locally without communication.

We also consider other models like asynchronous distributed shared-memory computing [5]. In this setting, every process has access to a global memory shared by all processes. Every process accesses this memory via atomic read and write instructions. The memory is composed of registers, and each process is allocated a set of private registers. Every process can read all the registers, but can only write in its own registers. Processes are given distinct identities in $[n] = \{1, \ldots, n\}$ for n-process systems. They runs asynchronously, and are subject to crashes. A process that crashes stops taking steps. An arbitrary large number of processes can crash. Hence, an algorithm must never include instructions leading a process to wait for actions by another process, as the latter process can crash. This model is thus often referred to as the WAIT-FREE model.

Finally, we briefly consider other models, including mobile computing [22], mostly in the fully-synchronous FSYNC model in graphs (where all mobile agents perform in lock-step, moving from nodes to adjacent nodes in a network), and distributed quantum computing (where processes have access to intricate variables).

2 Model and Definitions

Given a boolean predicate, a distributed decision algorithm is a distributed algorithm in which every process p must eventually output a value

$$\text{opinion}(p) \in \{\text{accept}, \text{reject}\}$$

such that the global system state satisfies the given predicate if and only if all processes accept. In other word, the global interpretation of the individual opinions produced by the processes is the logical conjunction of all these opinions:

$$\text{global verdict} = \bigwedge_{p} \text{opinion}(p).$$

Among the earliest references explicitly related to distributed decision, it is worth mentioning [1, 6, 42]. In this section, we describe the general framework of distributed decision, without explicit references to some specific underlying computational model.

The structure of the section is inspired from the structure of complexity classes in sequential complexity theory. Given the "base" class P of languages that are sequentially decidable by a Turing machine in time polynomial in the size of the input, the classes NP (for non-deterministic polynomial time) and BPP (for bounded probability polynomial time) are defined, as well as the classes Σ_k^{P} and Π_k^{P}, $k \geq 0$, of the polynomial hierarchy. In this section we assume given an abstract class BC (for *bounded distributed computing*), based on which larger classes can be defined. Such a base class BC could be a *complexity* class like, e.g., the class of graph properties that can be checked in constant time in the LOCAL model, or a *computability* class like, e.g., the class of system properties that can be checked in a shared-memory distributed system subject to crash failures. Given the "base" class BC, we shall define the classes NBC, BPBC, Σ_k^{BC} and Π_k^{BC}, that are to BC what NP, BPP, Σ_k^{P} and Π_k^{P} are to P, respectively.

2.1 Distributed Languages

A system *configuration* **C** is a (partial) description of a distributed system state. For instance, in distributed network computing, a configuration **C** is of the form (G, ℓ) where G is a graph, and $\ell : V(G) \to \{0,1\}^*$. Similarly, in shared memory computing, a configuration **C** is of the form $\ell : [n] \to \{0,1\}^*$ where n is the number of processes. The function ℓ is called *labeling* function, and $\ell(v)$ the *label* of v, which can be any arbitrary bit string. In the context of distributed decision, the label of a process is the input of that process.

For instance, the label of a node in a processor network can be a color, and the label of a process in a shared memory system can be a status like "elected" or "defeated". Note that, in both examples, a configuration is oblivious to the content of the shared memory and/or to the message in transit. The labeling function ℓ may not describe the full state of each process, but only the content of some specific variables.

Definition 1. *Given a distributed computing model, a* distributed language *is a Turing-computable set of configurations compatible with this model.*

For instance, in the framework of network computing,

$$\textsc{proper-coloring} = \{(G, \ell) : \forall \{u, v\} \in E(G), \ell(u) \neq \ell(v)\}$$

is the distributed language composed of all networks with a proper coloring of their nodes (the label $\ell(v)$ of node v is its color). Similarly, in the framework of crash-prone shared-memory computing,

$$\textsc{agreement} = \{\ell : \exists y \in \{0,1\}^*, \forall i \in [n], x(i) = y \text{ or } x(i) = \perp\}$$

is the distributed language composed of all systems where agreement between the non-crashed processes is achieved (the label of process p_i is $\ell(i)$, and the symbol $\perp$ refers to the scenario in which process p_i crashed).

For a fixed distributed language $\mathcal{L}$, a configuration in $\mathcal{L}$ is said to be *legal*, and a configuration not in $\mathcal{L}$ is said to be *illegal*. Any distributed language $\mathcal{L}$ defines a *construction* task, in which every process must compute a label such that the collection of labels outputted by the processes form a legal configuration for $\mathcal{L}$. In the following, we are mostly interested in *decision* tasks, where the labels of the nodes are given, and the processes must collectively check whether these labels form a legal configuration.

Notation. Given a system configuration $\mathbf{C}$ with respect to some distributed computing model, we denote by $V(\mathbf{C})$ the set of all computing entities (a.k.a. processes) in $\mathbf{C}$. This notation reflects the fact that, in the following, the set of processes will most often be identified as the vertex-set $V(G)$ of a graph G

2.2 Distributed Decision

Given a distributed computing model, let us define some *bounded computing* class BC as a class of distributed languages that can be decided with a distributed algorithm $\mathcal{A}$ using a bounded amount of resources. Such an algorithm $\mathcal{A}$ is said to be *bounded*. What is meant by "resource" depends on the computing model. In most of the models investigated in this paper, the resource of interest is the number of rounds (as in the LOCAL and $\mathsf{CONGEST}$ models), or the number of read/write operations (as in the $\mathsf{WAIT\text{-}FREE}$ model). A distributed language $\mathcal{L}$ is in BC if and only if there exists a bounded algorithm $\mathcal{A}$ such that, for any input configuration $\mathbf{C}$, the algorithm $\mathcal{A}$ outputs $\mathcal{A}(\mathbf{C}, v)$ at each process v, and this output satisfies:

$$\mathbf{C} \in \mathcal{L} \iff \text{for every } v \in V(\mathbf{C}),\ \mathcal{A}(\mathbf{C}, v) = \text{accept}. \tag{1}$$

That is, for every $\mathbf{C} \in \mathcal{L}$, running $\mathcal{A}$ on $\mathbf{C}$ results in all processes accepting $\mathbf{C}$. Instead, for every $\mathbf{C} \notin \mathcal{L}$, running $\mathcal{A}$ on $\mathbf{C}$ results in at least one process rejecting $\mathbf{C}$.

Example. In the context of network computing, PROPER-COLORING can be decided in one round, by having each node merely comparing its color with the ones of its neighbors, and accepting if and only if its color is different from all these colors. Similarly, in the context of shared-memory computing, AGREEMENT can be decided by having each node performing just one read/write operation, accepting if and only if all labels different from $\perp$ observed in memory are identical. In other words, assuming that BC is a network computing class bounding algorithms to perform in a constant number of rounds, we have

$$\text{PROPER-COLORING} \in \mathsf{BC}$$

for any model allowing each process to send its color to all its neighbors in a constant number of rounds, like, e.g., the LOCAL model. Similarly, assuming that BC is a shared-memory computing class bounding algorithms to perform in a constant number of read/write operations, we have

$$\text{agreement} \in \mathsf{BC}.$$

Notation. In the following, Eq. (1) will often be abbreviated with

$$\mathbf{C} \in \mathcal{L} \iff \mathcal{A}(\mathbf{C}) = \text{accept}$$

in the sense that $\mathcal{A}$ accepts if and only if each of the processes accepts.

Note that the rule of distributed decision, i.e., the logical conjunction of the individual boolean outputs of the processes is not symmetric. For instance, deciding whether a graph is properly colored can be done locally, while deciding whether a graph is *not* properly colored may require long-distance communications. On the other hand, asking for other rules, like unanimous decision (where all processes must reject an illegal configuration) or even just majority decision, would require long-distance communications for most classical decision problems.

2.3 Probabilistic Distributed Decision

The bounded computing class BC is a base class upon which other classes can be defined. Given $p, q \in [0, 1]$, we define the class $\mathsf{BPBC}(p, q)$, for *bounded probability bounded computing*, as the class of all distributed languages $\mathcal{L}$ for which there exists a randomized bounded algorithm $\mathcal{A}$ such that, for every configuration $\mathbf{C}$,

$$\begin{cases} \mathbf{C} \in \mathcal{L} & \Rightarrow & \Pr[\mathcal{A}(\mathbf{C}) = \text{accept}] \geq p; \\ \mathbf{C} \notin \mathcal{L} & \Rightarrow & \Pr[\mathcal{A}(\mathbf{C}) = \text{reject}] \geq q. \end{cases} \tag{2}$$

Such an algorithm $\mathcal{A}$ is called a (p, q)-decider for $\mathcal{L}$. Note that, as opposed to the class BPP of complexity theory, the parameters p and q are not arbitrary, in the sense that boosting the probability of success of a (p, q)-decider in order to get a (p', q')-decider with $p' > p$ and $q' > q$ is not always possible. Indeed, if $\mathcal{A}$ is repeated many times on an illegal instance, say k times, it may well be the case that each node will reject at most once during the k repetitions, because, at each iteration of $\mathcal{A}$, rejection could come from a different node. As a consequence, classical boosting techniques based on repetition and taking majority do not necessarily apply.

Example. Let us consider the following distributed language, where each process can be labeled either white or black, i.e., $\ell : V(\mathbf{C}) \to \{\circ, \bullet\}$:

$$\textsc{amos} = \{\ell : |\{v \in V(\mathbf{C}) : \ell(v) = \bullet\}| \leq 1\}.$$

Here, AMOS stands for "at most one selected", where a node v is selected if $\ell(v) = \bullet$. There is a trivial (p, q)-decider for AMOS as long as $p^2 + q \leq 1$, which works as follows. Every node v with $\ell(v) = \circ$ accepts (with probability 1). A node v with $\ell(v) = \bullet$ accepts with probability p, and rejects with probability $1 - p$. If $\mathbf{C} \in \textsc{amos}$, then Pr[all nodes accept $\mathbf{C}$] $\geq p$. If $\mathbf{C} \notin \textsc{amos}$, then Pr[at least one node rejects $\mathbf{C}$] $\geq 1 - p^2 \geq q$.

2.4 Distributed Verification

Given a bounded computing class BC, we describe the class NBC, which is to BC what NP is to P in complexity theory. We define the class NBC, for *non-deterministic bounded computing*, as the class of all distributed languages $\mathcal{L}$ such that there exists a bounded algorithm $\mathcal{A}$ satisfying that, for every configuration $\mathbf{C}$,

$$\mathbf{C} \in \mathcal{L} \iff \exists c : \mathcal{A}(\mathbf{C}, c) = \text{accepts} \tag{3}$$

where

$$c : V(\mathbf{C}) \to \{0, 1\}^*.$$

The function c is called the *certifying* function. It assigns a certificate to every process, and the certificates do not need to be identical. Note that the certificate $c(v)$ of process v must not be mistaken with the label $\ell(v)$ of that process.

The bounded algorithm $\mathcal{A}$ is also known as a *verification* algorithm for $\mathcal{L}$, as it verifies a given proof c, which is supposed to certify that $\mathbf{C} \in \mathcal{L}$. At each process $v \in V(\mathbf{C})$, the verification algorithm takes as input the pair $(\ell(v), c(v))$. Note that the appropriate certificate c leading to accept a configuration $\mathbf{C} \in \mathcal{L}$ may depend on the given configuration $\mathbf{C}$. However, for $\mathbf{C} \notin \mathcal{L}$, the verification algorithm $\mathcal{A}$ must systematically guaranty that at least one process rejects, whatever the given certificate function is.

Alternatively, one can interpret Eq. (3) as a game between a *prover* which, for every configuration $\mathbf{C}$, assigns a certificate $c(v)$ to each process $v \in V(\mathbf{C})$, and a *verifier* which checks that the certificates assigned by the prover collectively form a *proof* that $\mathbf{C} \in \mathcal{L}$. For a legal configuration (i.e., a configuration in $\mathcal{L}$) the prover must be able to produce a distributed proof leading the distributed verifier to accept, while, for an illegal configuration, the verifier must reject in at least one node whatever the proof provided by the prover is.

Example. Let us consider the distributed language

$$\textsc{acyclic} = \{(G, \ell) : G \text{ has no cycles}\}$$

in the context of network computing. Note that ACYCLIC cannot be decided locally, even in the LOCAL model. However, ACYCLIC can be verified in just one round. If G is acyclic, i.e., G is a forest, then let us select an arbitrary node in each tree of G, and call it a root. Next, let us assign to each node $u \in V(G)$ the certificate $c(u)$ equal to its distance to the root of its tree. The verification algorithm $\mathcal{A}$ then proceeds at every node u as follows. Node u exchanges its certificate with the ones of it neighbors, and checks that it has a unique neighbor v satisfying $c(v) = c(u) - 1$, and all the other neighbors $w \neq v$ satisfying $c(w) = c(u) + 1$. (If u has $c(u) = 0$, then it checks that all its neighbors w have $c(w) = 1$). If all tests are passed, then u accepts, else it rejects. If G is a acyclic, then, by construction, the verification accepts at all nodes. Instead, if G has a cycle, then, for every setting of the certifying function, some inconsistency will be detected by at least one node of the cycle, which leads this node to reject. Hence

$$\textsc{acyclic} \in \mathsf{NBC}$$

where BC bounds the number of rounds, for every distributed computing model allowing every node to exchange $O(\log n)$ bits along each of its incident edges at every round, like, e.g., the $\mathsf{CONGEST}$ model.

Notation. For any function $f : \mathbb{N} \to \mathbb{N}$, we define $\mathsf{NBC}(f)$ as the class NBC where the certificates are bounded to be on at most $f(n)$ bits in n-node networks. For $f \in \Theta(\log n)$, $\mathsf{NBC}(f)$ is rather denoted by log-NBC.

2.5 Distributed Decision Hierarchy

In the same way the polynomial hierarchy PH is built upon P using alternating universal and existential quantifiers, one can define a hierarchy built upon base class BC. Given a class BC for some distributed computing model, we define the *distributed decision hierarchy* $\mathsf{DH}^{\mathsf{BC}}$ as follows. We set $\Sigma_0^{\mathsf{BC}} = \Pi_0^{\mathsf{BC}} = \mathsf{BC}$, and, for $k \geq 1$, we set Σ_k^{BC} as the class of all distributed languages $\mathcal{L}$ such that there exists a bounded algorithm $\mathcal{A}$ satisfying that, for every configuration $\mathbf{C}$,

$$\mathbf{C} \in \mathcal{L} \iff \exists c_1 \, \forall c_2 \, \exists c_3 \ldots Q c_k : \mathcal{A}(\mathbf{C}, c_1, \ldots, c_k) = \text{accept}$$

where, for every $i \in \{1, \ldots, k\}$, $c_i : V(\mathbf{C}) \to \{0,1\}^*$, and Q is the universal quantifier if k is even, and the existential one otherwise. The class Π_k^{BC} is defined similarly, by having a universal quantifier as first quantifier, as opposed to an existential one as in Σ_k^{BC}. The c_i's are called *certifying* functions. In particular, we have

$$\mathsf{NBC} = \Sigma_1^{\mathsf{BC}}.$$

Finally, we define

$$\mathsf{DH}^{\mathsf{BC}} = (\cup_{k\geq 0} \Sigma_k^{\mathsf{BC}}) \cup (\cup_{k\geq 0} \Pi_k^{\mathsf{BC}}).$$

As for NBC, a class Σ_k^{BC} or Π_k^{BC} can be viewed as a game between a prover (playing the existential quantifiers), a disprover (playing the universal quantifiers), and a verifier (running a verification algorithm $\mathcal{A}$).

Example. Let us consider the distributed language

$$\textsc{vertex-cover} = \{(G, \ell) : \{v \in V(G) : \ell(v) = 1\} \text{ is a minimum vertex cover}\}$$

in the context of network computing. We show that $\textsc{vertex-cover} \in \Pi_2^{\mathsf{BC}}$, that is, there exists a bounded distributed algorithm $\mathcal{A}$ such that

$$(G, \ell) \in \textsc{vertex-cover} \iff \forall c_1\, \exists c_2 : \mathcal{A}(G, \ell, c_1, c_2) = \text{accept}$$

where BC is any network computing class bounding algorithms to perform in a constant number of rounds. For any configuration (G, ℓ), the disprover tries to provide a vertex cover $c_1 : V(G) \to \{0, 1\}$ of size smaller than the solution ℓ, i.e., $|\{v \in V(G) : c_1(v) = 1\}| < |\{v \in V(G) : \ell(v) = 1\}|$. On a legal configuration (G, ℓ), the prover then reacts by providing each node v with a certificates $c_2(v)$ such that the c_2-certificates collectively encode a spanning tree (and its proof) aiming at demonstrating that there is an error in c_1 (like c_1 is actually not smaller than ℓ, or c_1 is not covering some edge, etc.). It follows that

$$\textsc{vertex-cover} \in \Pi_2^{\mathsf{BC}}$$

for any model allowing each process to exchange $O(\log n)$-bits messages with its neighbors in a constant number of rounds, like, e.g., the CONGEST model.

Notation. Similarly to the class NBC, for any function $f : \mathbb{N} \to \mathbb{N}$, we define $\Sigma_k^{\mathsf{BC}}(f)$ (resp., $\Pi_k^{\mathsf{BC}}(f)$) as the class Σ_k^{BC} (resp., Π_k^{BC}) where all certificates are bounded to be on at most $f(n)$ bits in n-node networks. For $f \in \Theta(\log n)$, these classes are denoted by log-Σ_k^{BC} and log-Π_k^{BC}, respectively. The classes $\mathsf{DH}^{\mathsf{BC}}(f)$ and log-$\mathsf{DH}^{\mathsf{BC}}$ are defined similarly.

3 Distributed Decision in Networks

In this section, we focus on languages defined as collections of configurations of the form (G, ℓ) where G is a simple connected n-node graph, and $\ell : V(G) \to \{0, 1\}^*$ is a labeling function assigning to every node v a label $\ell(v)$. Recall that an algorithm $\mathcal{A}$ is *deciding* a distributed language $\mathcal{L}$ if and only if, for every configuration (G, ℓ),

$$(G, \ell) \in \mathcal{L} \iff \mathcal{A}(G, \ell) \text{ accepts at all nodes.}$$

3.1 LOCAL model

3.1.1 Local Distributed Decision (LD and BPLD)

In their seminal paper [48], Naor and Stockmeyer define the class LCL, for *locally checkable labelings*. Let $\Delta \geq 0$, $k \geq 0$, and $t \geq 0$, and let $\mathcal{B}$ be a set of balls of radius at most t with nodes of degree at most Δ, labeled by labels in $[k]$. Note that $\mathcal{B}$ is finite. Such a set $\mathcal{B}$ defines the language $\mathcal{L}$ consisting of all configurations (G, ℓ) where G is a graph with maximum degree Δ, and $\ell : V(G) \to [k]$, such that all balls of radius t in (G, ℓ) belong to $\mathcal{B}$. The set $\mathcal{B}$ is called the set of *good* balls for $\mathcal{L}$. LCL is the class of languages that can be defined by a set of good balls, for some parameters Δ, k, and t. For instance the set of k-colored graphs with maximum degree Δ is a language in LCL. The good balls of this LCL language are simply the balls of radius 1 where the center node is labeled with a color different from all the colors of its neighbors.

A series of results were achieved in [48] about LCL languages. In particular, it is Turing-undecidable whether any given $\mathcal{L} \in$ LCL has a construction algorithm running in $O(1)$ rounds in the LOCAL model. Also, [48] showed that the node IDs play a limited role in the context of LCL languages. Specifically, [48] proves that, for every $r \geq 0$, if a language $\mathcal{L} \in$ LCL has a r-round construction algorithm, then it has also a r-round *order invariant* construction algorithm, where an algorithm is order invariant if the *relative order* of the node IDs may play a role, but not the actual *values* of these IDs. The assumption $\mathcal{L} \in$ LCL can actually be discarded, as long as $\mathcal{L}$ remains defined on constant degree graphs with constant labels. That is, [3] proved that, in constant degree graphs, if a language with constant size labels has a r-round construction algorithm, then it has also a r-round order invariant construction algorithm. Last but not least, [48] established that randomization is of little help in the context of LCL languages. Specifically, [48] proves that if a language $\mathcal{L} \in$ LCL has a randomized Monte-Carlo construction algorithm running in $O(1)$ rounds, then $\mathcal{L}$ also has a deterministic construction algorithm running in $O(1)$ rounds.

The class LD, for *local decision* was defined in [33] as the class of all distributed languages that can be decided in $O(1)$ rounds in the LOCAL model. The class LD is the basic class playing the role of BC in the context of local decision. Hence LCL $\subseteq$ LD since the set of good balls of a language in LCL is, by definition, finite. On the other hand, LCL $\subset$ LD, where the inclusion is strict since LD does not restrict the graphs to be of bounded degree, nor the labels to be of bounded size. Given $p, q \in [0, 1]$, the class BPLD(p, q), for *bounded probability local decision*, was defined in [33] as the class of languages for which there is a (p, q)-decider running in $O(1)$ rounds in the LOCAL model. For $p^2 + q \leq 1$, BPLD(p, q) is shown to include languages that cannot be even decided deterministically in $o(n)$

rounds. On the other hand, [33] also establishes a derandomization result, stating that, for $p^2 + q > 1$, if $\mathcal{L} \in \mathsf{BPLD}(p, q)$, then $\mathcal{L} \in \mathsf{LD}$. This results however holds only for languages closed under node deletion, and it is proved in [27] that, for any every $c \geq 2$, there exists a language $\mathcal{L}$ with a (p, q)-decider satisfying $p^c + q > 1$ and running in a single round, which cannot be decided deterministically in $o(\sqrt{n})$ rounds. On the other hand, [27] proves that, for $p^2 + q > 1$, we have $\mathsf{BPLD}(p, q) = \mathsf{LD}$ for all languages restricted on paths.

On the negative side, it was proved in [27] that boosting the probability of success for decision tasks is not always achievable in the distributed setting, by considering the classes

$$\mathsf{BPLD}_k = \bigcup_{p^{1+1/k}+q>1} \mathsf{BPLD}(p, q) \text{ and } \mathsf{BPLD}_\infty = \bigcup_{p+q>1} \mathsf{BPLD}(p, q)$$

for any $k \geq 1$, and proving that, for every $k \geq 1$, $\mathsf{BPLD}_k \subset \mathsf{BPLD}_\infty$, and $\mathsf{BPLD}_k \subset \mathsf{BPLD}_{k+1}$, where all inclusions are strict.

On the positive side, it was proved in [20] that the result in [48] regarding the derandomization of construction algorithms can be generalized from LCL to BPLD. Namely, [20] proves that, for languages on bounded degree graphs and bounded size labels, for every $p > \frac{1}{2}$ and $q > \frac{1}{2}$, if $\mathcal{L} \in \mathsf{BPLD}(p, q)$ has a randomized Monte-Carlo construction algorithm running in $O(1)$ rounds, then $\mathcal{L}$ has also a deterministic construction algorithm running in $O(1)$ rounds.

3.1.2 Identity-Oblivious Algorithms (LDO)

In the LOCAL model, a distributed algorithm is *identity-oblivious*, or simply *ID-oblivious*, if the outputs of the nodes are not impacted by the identities assigned to the nodes. That is, for any two ID-assignments given to the nodes, the output of every node must be identical in both cases. Note that an *identity-oblivious* algorithm may use the IDs of the nodes (e.g., to distinguish them), but the output must be oblivious to these IDs.

The class LDO, for *local decision oblivious* was defined in [28, 29], as the class of all distributed languages that can be decided in $O(1)$ rounds by an ID-oblivious algorithm in the LOCAL model. The class LDO is the basic class playing the role of BC in the context of ID-oblivious local decision. It is shown in [29] that $\mathsf{LDO} = \mathsf{LD}$ when restricted to languages that are closed under node deletion. However, it is proved in [28] that $\mathsf{LDO} \subset \mathsf{LD}$, where the inclusion is strict. In the language $\mathcal{L} \in \mathsf{LD} \setminus \mathsf{LDO}$ used in [28] to prove the strict inclusion $\mathsf{LDO} \subset \mathsf{LD}$, each node label includes a Turing machine M. Establishing $\mathcal{L} \in \mathsf{LD}$ makes use of an algorithm simulating M at each node, for a number of rounds equal to the identity of the node. Establishing $\mathcal{L} \notin \mathsf{LDO}$ makes use of the fact that an ID-oblivious algorithm can be sequentially simulated, and therefore, if an ID-oblivious algorithm would allow to decide $\mathcal{L}$, then by simulation of this algorithm, there would exist a sequential algorithm for separating the set of Turing machines that halts and output 0 from the set of Turing machines that halts and output 1, which is impossible.

In [29,30], the power of IDs in local decision is characterized using *oracles*. An oracle is a trustable party with full knowledge of the input, who can provide nodes with information about this input. It is shown in [29] that $\mathsf{LDO} \subseteq \mathsf{LD} \subseteq \mathsf{LDO}^{\#\text{node}}$ where #node is the oracle providing each node with an arbitrary large upper bound on the number of nodes. A *scalar* oracle f returns a list $f(n) = (f_1, \ldots, f_n)$ of n values that are assigned arbitrarily to the n nodes in a one-to-one manner. A scalar oracle f is large if, for any set of k nodes, the largest value provided by f to the nodes in this set grows with k. [30] proved that, for any computable scalar oracle f, we have $\mathsf{LDO}^f = \mathsf{LD}^f$ if and only if f is large, where LD^f (resp., LDO^f) is the class of languages that can be locally decided in $O(1)$ rounds in the LOCAL model by an algorithm (resp., by an ID-oblivious algorithm) which uses the information provided by f available at the nodes.

3.1.3 Anonymous Networks

Derandomization results were achieved in [19] in the framework of anonymous network (that is, nodes have no IDs). Namely, for every language $\mathcal{L}$ that can be decided locally in any anonymous network, if there exists a randomized anonymous construction algorithm for $\mathcal{L}$, then there exists a deterministic anonymous construction algorithm for $\mathcal{L}$, provided that the latter is equipped with a 2-hop coloring of the input network.

3.2 CONGEST model

3.2.1 Non-Local Algorithms

In [44] and [17] the authors consider decision problems such as checking whether a given set of edges forms a spanning tree, checking whether a given set of edges forms a minimum-weight spanning tree (MST), checking various forms of connectivity, etc. All these decision tasks require essentially $\Theta(\sqrt{n} + D)$ rounds (the lower bound is typically obtained using reduction to communication complexity). In particular, [17] proved that checking whether a given set of edges is a spanning tree requires $\Omega(\sqrt{n} + D)$ rounds, which is much more that what is required to construct a spanning tree ($O(D)$ rounds, using a simple breadth-first search). However, [17] proved that, for some other problems (e.g., MST), lower bounds on the round-complexity of the decision task consisting in checking whether a solution is valid yield lower bounds on the round-complexity of the corresponding construction task, and this holds also for the construction of approximate solutions.

The *congested clique* model is the CONGEST model restricted to complete graphs. Deciding whether a graph given as input contains some specific patterns as subgraphs has been considered in [16] and [18] for the congested clique. In particular, [16] provides an algorithm for deciding the presence of a k-node cycle C_k running in $O(2^{O(k)}n^{0.158})$-rounds.

3.2.2 Local Algorithms

Very few distributed languages on graphs can be checked locally in the CONGEST model. For instance, even just deciding whether G contains a triangle cannot be done in $O(1)$ rounds in the CONGEST model. *Distributed property testing* is a framework recently introduced in [15]. Let $0 < \epsilon < 1$ be a fixed parameter. Recall that, according to the usual definition borrowed from *property testing* (in the so-called *sparse* model), a graph property P is ϵ-far from being satisfied by an m-edge graph G if applying a sequence of at most ϵm edge-deletions or edge-additions to G cannot result in a graph satisfying P. We say that a distributed algorithm $\mathcal{A}$ is a distributed *testing* algorithm for P if and only if, for any graph G modeling the actual network,

$$\begin{cases} G \text{ satisfies } P \implies \Pr[\mathcal{A} \text{ accepts } G \text{ in all nodes}] \geq \frac{2}{3}; \\ G \text{ is } \epsilon\text{-far from satisfying } P \implies \Pr[\mathcal{A} \text{ rejects } G \text{ in at least one node}] \geq \frac{2}{3}. \end{cases}$$

Among other results, [15] proved that, in bounded degree graphs, bipartiteness can be distributedly tested in $O(\text{polylog}\, n)$ rounds in the CONGEST model. Moreover, it is also proved that triangle-freeness can be distributedly tested in $O(1)$ rounds. (The dependence in ϵ is hidden in the big-O notation). This latter result has been recently extended in [40] to testing H-freeness, for every 4-node graph H, in $O(1)$ rounds. On the other hand, it is not known whether distributed testing K_5-freeness or C_5-freeness can be achieved in $O(1)$ rounds, and [40] proves that "natural" approaches based on DFS or BFS traversals do not work.

3.3 General Interpretation of Individual Outputs

In [3, 4], a generalization of distributed decision is considered, where every node outputs not just a single bit (accept or reject), but can output an arbitrary bit-string. The global verdict is then taken based on the multi-set of all the binary strings outputted by the nodes. The concern is restricted to decision algorithms performing in $O(1)$ rounds in the LOCAL model, and the objective is to minimize the size of the outputs. The corresponding basic class BC for outputs on $O(1)$ bits is denoted by ULD, for *universal* LD. (It is universal in the sense that the global interpretation of the individual outputs is not restricted to the logical conjunction). It is proved in [3] that, for any positive even integer Δ, every distributed decision algorithm for cycle-freeness in connected graphs with degree at most Δ must produce outputs of size at least $\lceil \log \Delta \rceil - 1$ bits. Hence, cycle-freeness does not belong to ULD in general, but it does belong to ULD for constant degree graphs.

In [11] the authors consider a model in which each node initially knows the IDs of its neighbors, while the nodes do not communicate through the edges of the network but via a public whiteboard. The concern of [11] is mostly restricted to the case in which every node can write only once on the whiteboard, and the objective is to minimize the size of the message written by each node on the whiteboard. The global verdict is then taken based on the collection of messages written on the whiteboard. It is shown that, with just $O(\log n)$-bit messages, it is possible to rebuild the whole graph from the information on the whiteboard as long as the graph is planar or, more generally, excluding a fixed minor. Variants of the model are also considered, in which problems such as deciding triangle-freeness or connectivity are considered. See also [43] for deciding the presence of induced subgraphs.

4 Distributed Verification in Networks

In this section, we still focus on languages defined as collections of configurations of the form (G, ℓ) where G is a simple connected n-node graph, and $\ell : V(G) \to \{0,1\}^*$ is a labeling function. Recall that an algorithm $\mathcal{A}$ is *verifying* a distributed language $\mathcal{L}$ if and only if, for every configuration (G, ℓ),

$$(G,\ell) \in \mathcal{L} \iff \exists c : \mathcal{A}(G,\ell,c) \text{ accepts at all nodes} \tag{4}$$

where $c : V(G) \to \{0,1\}^*$, and $c(v)$ is called the *certificate* of node $v \in V(G)$. Again, the certificate $c(v)$ of node v must not be mistaken with the label $\ell(v)$ of node v. Also, the notion of certificate must not be confused with the notion of *advice*. While the latter are trustable information provided by an oracle [26,31,32], the former are proofs that must be verified.

We survey the results about the class NBC $= \Sigma_1^{\mathsf{BC}}$ where the basic class BC is LD, LDO, ULD, etc.

4.1 LOCAL model

It is crucial to distinguish two cases in Eq. (4), depending on whether the certificates can depend on the identities assigned to the nodes, or not, as reflected in Eq. (5) and (6) below.

4.1.1 Local Distributed Verification (Σ_1^{LD}, PLS, and LCP)

A distributed language $\mathcal{L}$ satisfies $\mathcal{L} \in \Sigma_1^{\mathsf{LD}}$ if and only if there exists a verification algorithm $\mathcal{A}$ running in $O(1)$ rounds in the LOCAL model such that, for every configuration (G, ℓ), we have

$$\begin{cases} (G,\ell) \in \mathcal{L} & \Rightarrow \quad \forall \mathrm{ID}, \exists c,\ \mathcal{A}(G,\ell,c) \text{ accepts at all nodes} \\ (G,\ell) \notin \mathcal{L} & \Rightarrow \quad \forall \mathrm{ID}, \forall c,\ \mathcal{A}(G,\ell,c) \text{ rejects in at least one node} \end{cases} \tag{5}$$

where $c : V(G) \to \{0,1\}^*$, and where, for $(G,\ell) \in \mathcal{L}$, the assignment of the certificates to the nodes may depend on the identities given to these nodes. This notion has actually been introduced under the terminology *proof-labeling scheme* in [47], where the concern is restricted to verification algorithms running in just a single round, with the objective of minimizing the size of the certificates. In particular, it is proved that minimum-weight spanning tree can be verified with certificates on $O(\log^2)$ bits in n-node networks, and this bound in tight [45] (see also [44]). Interestingly, the $\Omega(\log^2 n)$ bits lower bound on the certificate size can be broken, and reduced to $O(\log n)$ bits, to the price of allowing verification to proceed in $O(\log n)$ rounds [46]. There are tight connections between proof-labeling schemes and compact silent self-stabilizing algorithms [13], and proof-labeling schemes can even be used as a basis to semi-automatically derive compact time-efficient self-stabilizing algorithms [12]. Let PLS be the class of distributed languages for which there exists a proof-labeling scheme. We have

$$\mathsf{PLS} = \mathsf{ALL}$$

where ALL is the class of all distributed languages on networks (i.e., with configurations of the form (G,ℓ)). This equality is however achieved using certificates on $O(n^2 + nk)$ bits in n-node networks, where k is the maximum size of the labels in the given configuration (G,ℓ). The $O(n^2)$ bits are used to encode the adjacency matrix of the network, and the $O(nk)$ bits are used to encode the inputs to the nodes.

The notion of proof-labeling scheme has been extended in [41] to the notion of *locally checkable proofs*, which is the same as proof-labeling scheme but where the verification algorithm is not bounded to run in a single round, but may perform an arbitrarily large constant number of rounds. Let LCP be the associate class of distributed languages. By definition, we have

$$\mathsf{LCP} = \Sigma_1^{\mathsf{LD}},$$

and, more specifically,

$$\mathsf{LCP}(f) = \Sigma_1^{\mathsf{LD}}(f)$$

for every function f bounding the size of the certificates. Moreover, since PLS = ALL, it follows that

$$\mathsf{PLS} = \mathsf{LCP} = \Sigma_1^{\mathsf{LD}} = \mathsf{ALL}.$$

Yet, allowing more rounds for the verification may enable to save space in the certificate size. This is indeed the case for some languages [10], that is there are functions f for which

$$\mathsf{PLS}(f) \subset \mathsf{LCP}(f)$$

with strict inclusions. It is proved in [41] that there are natural languages (e.g., the set of graphs with a non-trivial automorphism, 3-non-colorability, etc.) which require certificates on $\tilde{\Omega}(n^2)$ bits in n-node networks. Recently, [9] introduced a mechanism enabling to reduce exponentially the amount of communication in proof-labeling schemes, using randomization. See also [51] for applications of locally checkable proofs to software-defined networks.

4.1.2 Identity-Oblivious Algorithms (Σ_1^{LDO} and NLD)

A distributed language $\mathcal{L}$ satisfies $\mathcal{L} \in \Sigma_1^{\mathsf{LDO}}$ if and only if there exists a verification algorithm $\mathcal{A}$ running in $O(1)$ rounds in the LOCAL model such that, for every configuration (G, ℓ), we have

$$\begin{cases} (G,\ell) \in \mathcal{L} & \Rightarrow \quad \exists c,\ \forall \mathrm{ID},\ \mathcal{A}(G,\ell,c) \text{ accepts at all nodes} \\ (G,\ell) \notin \mathcal{L} & \Rightarrow \quad \forall c,\ \forall \mathrm{ID},\ \mathcal{A}(G,\ell,c) \text{ rejects in at least one node} \end{cases} \tag{6}$$

where $c : V(G) \to \{0,1\}^*$, and, for $(G,\ell) \in \mathcal{L}$, the assignment of the certificates to the nodes must not depend on the identities given to these nodes. In [33], the class NLD, for *non-deterministic local decision* is introduced. In NLD, even if the certificates must not depend on the identities of the nodes, the verification algorithm is not necessarily identity-oblivious. Yet, it was proved in [29] that restricting the verification algorithm to be identity-oblivious does not restrict the power of the verifier. Hence,

$$\mathsf{NLD} = \Sigma_1^{\mathsf{LDO}}$$

Σ_1^{LDO} is characterized in [29] as the class of languages that are *closed under lift*, where H is a k-lift of G if there exists an homomorphism from H to G preserving radius-k balls. Hence,

$$\Sigma_1^{\mathsf{LDO}} \subset \mathsf{ALL}$$

where the inclusion is strict. However, it was proved in [33] that, for every distributed language $\mathcal{L}$, and for every p, q such that $p^2 + q \leq 1$, there is a non-deterministic (p, q)-decider for $\mathcal{L}$. In other words, for every p, q such that $p^2+q \leq 1$, we have

$$\mathsf{BPNLD}(p,q) = \mathsf{ALL}.$$

In [33], a complete problem for NLD was identified. However, it was recently noticed in [7] that the notion of local reduction used in [33] is way too strong, enabling to bring languages outside NLD into NLD. A weaker notion of local reduction was thus defined in [7], preserving the class NLD. A language is proved to be NLD-complete under this weaker type of local reduction.

4.1.3 Anonymous Networks

Distributed verification in the context of fully anonymous networks (no node-identities, and no port-numbers) has been considered in [23].

4.2 CONGEST model (log-Σ_1^{LD} and log-LCP)

The class log-LCP, that is, log-Σ_1^{LD}, i.e., Σ_1^{LD} with certificates of size $O(\log n)$ bits, was investigated in [41]. This class fits well with the CONGEST model, which allows to exchange messages of at most $O(\log n)$ bits at each round. For instance, non-bipartiteness is in log-LCP. Also, restricted to bounded-degree graphs, there are problems in log-LCP that are not contained in NP, but log-LCP $\subseteq$ NP/poly, i.e., NP with a polynomial-size non-uniform advice. Last but not least, [41] shows that existential MSO on connected graphs is included in log-LCP.

4.3 General Interpretation of Individual Outputs

As already mentioned in Section 3.3, a generalization of distributed decision was considered in [3,4], where every node outputs not just a single bit (accept or reject), but can output an arbitrary bit-string. The global verdict is then taken based on the multi-set of all the binary strings outputted by the nodes. The concern is restricted to decision algorithm performing in $O(1)$ rounds in the LOCAL model, and the objective is to minimize the size of the output. The certificates must not depend on the node IDs, that is, verification proceed as specified in Eq. (6). For constant size outputs, it is shown in [4] that the class UNLD = Σ_1^{ULD} satisfies

$$\mathsf{UNLD} = \mathsf{ALL}$$

with just 2-bit-per-node outputs, which has to be consider in contrast to the fact that NLD is restricted to languages that are closed under lift (cf. Section 4.1.2). This result requires using certificates on $O(n^2 + nk)$ bits in n-node networks, where k is the maximum size of the labels in the given configuration (G, ℓ), but [4] shows that this is unavoidable. Also, while verifying cycle-freeness using the logical conjunction of the 1-bit-per-node outputs requires certificates on $\Omega(\log n)$ bits [41], it is proved in [4] that, by simply using the conjunction and the disjunction operators together, on only 2-bit-per-node outputs, one can verify cycle-freeness using certificates of size $O(1)$ bits.

5 Local Hierarchies in Networks

In this section, we survey the results about the hierarchies Σ_k^{BC} and Π_k^{BC}, $k \geq 0$, for different basic classes BC, including LD, LDO, etc.

5.1 LOCAL model ($\mathsf{DH}^{\mathsf{LD}}$ and $\mathsf{DH}^{\mathsf{LDO}}$)

We have seen in Section 4.1.1 that $\Sigma_1^{\mathsf{LD}} = \mathsf{ALL}$, which implies that the local distributed hierarchy $\mathsf{DH}^{\mathsf{LD}}$ collapses at the first level. On the other hand, we have also seen in Section 4.1.2 that $\Sigma_1^{\mathsf{LDO}} \subset \mathsf{ALL}$, where the inclusion is strict as Σ_1^{LDO} is restricted to languages that are closed under lift. It was recently proved in [7] that

$$\mathsf{LDO} \subset \Pi_1^{\mathsf{LDO}} \subset \Sigma_1^{\mathsf{LDO}} = \Sigma_2^{\mathsf{LDO}} \subset \Pi_2^{\mathsf{LDO}} = \mathsf{ALL}$$

where all inclusions are strict. Hence, the local ID-oblivious distributed hierarchy collapses at the second level. Moreover, it is shown that Π_2^{LDO} has a complete problem for local label-preserving reductions. (A complete problem for ALL was also identified in [33], but using an inappropriate notion of local reduction).

In the context of a general interpretation of individual outputs (see Section 4.3), [4] proved that $\Sigma_1^{\mathsf{ULD}} = \mathsf{ALL}$.

5.2 CONGEST model (log-$\mathsf{DH}^{\mathsf{LD}}$)

We have previously seen that $\Sigma_1^{\mathsf{LD}} = \mathsf{ALL}$. However, this requires certificates of polynomial size. In order to fit with the constraints of the $\mathsf{CONGEST}$ model, the local distributed hierarchy with certificate of logarithmic size was recently investigated in [21]. While it follows from [45] that $\textsc{mst} \notin \text{log-}\Sigma_1^{\mathsf{LD}}$, it is shown in [21] that

$$\textsc{mst} \in \text{log-}\Pi_2^{\mathsf{LD}}.$$

In fact, [21] proved that, for any $k \geq 1$,

$$\text{log-}\Sigma_{2k}^{\mathsf{LD}} = \text{log-}\Sigma_{2k-1}^{\mathsf{LD}} \text{ and } \text{log-}\Pi_{2k+1}^{\mathsf{LD}} = \text{log-}\Pi_{2k}^{\mathsf{LD}},$$

and thus focused only on the hierarchy $(\Lambda_k)_{k\geq 0}$ defined by $\Lambda_0 = \mathsf{LD}$, and, for $k \geq 1$,

$$\Lambda_k = \begin{cases} \text{log-}\Sigma_k^{\mathsf{LD}} & \text{if } k \text{ is odd} \\ \text{log-}\Pi_k^{\mathsf{LD}} & \text{if } k \text{ is even.} \end{cases}$$

It is proved that if there exists $k \geq 0$ such that $\Lambda_{k+1} = \Lambda_k$, then $\Lambda_{k'} = \Lambda_k$ for all $k' \geq k$. That is, the hierarchy collapses at the k-th level. Moreover, there exists a distributed language on 0/1-labelled oriented paths that is outside the Λ_k-hierarchy, and thus outside log-$\mathsf{DH}^{\mathsf{LD}}$. However, deciding whether a given solution to several optimisation problems such as maximum independent set, minimum dominating set, maximum matching, max-cut, min-cut, traveling salesman, etc., is optimal are all in co-Λ_1, and thus in log-Π_2^{LD}. The absence of a non-trivial automorphism is proved to be in Λ_3, that is log-Σ_3^{LD} — recall that this language requires certificated of $\tilde{\Omega}(n^2)$ bits to be placed in Σ_1^{LD} (see [41]). It is however not known whether $\Lambda_3 \neq \Lambda_2$, that is whether log-$\Pi_2^{\mathsf{LD}} \subset$ log-Σ_3^{LD} with a strict inclusion.

5.3 Distributed Graph Automata ($\mathsf{DH}^{\mathsf{DGA}}$)

An analogue of the polynomial hierarchy, where sequential polynomial-time computation is replaced by distributed local computation was recently investigated in [50]. The model in [50] is called *distributed graph automata*. This model assumes a finite-state automaton at each node (instead of a Turing machine), and assumes anonymous computation (instead of the presence of unique node identities). Also, the model assumes an arbitrary interpretation of the outputs produced by each automaton, based on an arbitrary mapping from the collection of all automata states to $\{\text{true}, \text{false}\}$. The main result in [50] is that the hierarchy $\mathsf{DH}^{\mathsf{DGA}}$ coincides with MSO on graphs.

6 Other Computational Models

6.1 Wait-Free Computing

The class WFD defined as the class of all distributed languages that are wait-free decidable was characterized in [36] as the class of languages satisfying the so-called *projection-closeness* property. For non projection-closed languages, [37] investigated more general interpretation of the individual opinions produced by the processes, beyond the logical conjunction of boolean opinons. In [35], it is proved that k-set agreement requires that the processes must be allowed to produce essentially k different opinions to be wait-free decided. The class Σ_1^{WFD} has been investigated in [38, 39], with applications to the space complexity of failure detectors. Interestingly, it is proved in [14] that wait-free decision finds applications to run-time verification.

6.2 Mobile Computing

The class MAD, for *mobile agent decision* has been considered in [34], as well as the class $\mathsf{MAV} = \Sigma_1^{\mathsf{MAD}}$, for *mobile agent verification*. It is proved that MAV has a complete language for a basic notion of reduction. The complement classes of

MAD and MAV have been recently investigated in [8] together with sister classes defined by other ways of interpreting the opinions of the mobile agents.

6.3 Quantum Computing

Distributed decision in a framework in which nodes can have access to extra ressources, such as shared randomness, or intricate variables (in the context of quantum computing) is discussed in [2].

7 Conclusion

Distributed decision and and distributed verification are known to have applications to very different contexts of distributed computing, including self-stabilization, randomized algorithms, fault-tolerance, runtime verification, etc. In this paper, our aim was to survey the results targeting distributed decision and verification per se. Beside the many interesting problems left open in each of the references listed in this paper, we want to mention two important issues.

Lower bounds in decision problems are often based on spatial or temporal arguments. Typically, the lack of information about far away processes, or the lack of information about desynchronized (or potentially crashed) processes, prevents processes to forge a consistent opinion about the global status of the distributed system. In the context of shared ressources, such type of arguments appears however to be too weak (cf. [2]). Similarly, lower bounds in verification problems are often based on reduction to communication complexity theory. However, such reductions appear to be difficult to apply to higher classes in the local hierarchy, like separating the class at the third level from the class at the second level of the local hierarchy with $O(\log n)$-bit certificates (cf. [21]).

This paper has adopted a systematic approach for presenting the results related to distributed decision and verification from the literature. This approach was inspired from sequential complexity and sequential computability theories. Such an approach provides a framework that enables to clearly separate decision from verification, as well as clearly separate the results obtained under different assumption (ID-oblivious, size of certificates, etc.). As already mentioned in [24], we believe that distributed decision provides a framework in which bridges between very different models might be identified, as decision tasks enables easy reductions between languages, while construction tasks are harder to manipulate because of the very different natures of their outputs.

References

[1] Yehuda Afek, Shay Kutten, and Moti Yung. The local detection paradigm and its application to self-stabilization. *Theor. Comput. Sci.*, 186(1-2):199–229, 1997.

[2] Heger Arfaoui and Pierre Fraigniaud. What can be computed without communications? *SIGACT News*, 45(3):82–104, 2014.

[3] Heger Arfaoui, Pierre Fraigniaud, David Ilcinkas, and Fabien Mathieu. Distributedly testing cycle-freeness. In *40th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 15–28, 2014.

[4] Heger Arfaoui, Pierre Fraigniaud, and Andrzej Pelc. Local decision and verification with bounded-size outputs. In *15th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 133–147, 2013.

[5] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics*. Wiley, 2004.

[6] Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction (extended abstract). In *32nd Symposium on Foundations of Computer Science (FOCS)*, pages 268–277, 1991.

[7] Alkida Balliu, Gianlorenzo D'Angelo, Pierre Fraigniaud, and Dennis Olivetti. Local distributed verification. *CoRR*, abs/1605.03892, 2016.

[8] Evangelos Bampas and David Ilcinkas. On mobile agent verifiable problems. In *12th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 123–137, 2016.

[9] Mor Baruch, Pierre Fraigniaud, and Boaz Patt-Shamir. Randomized proof-labeling schemes. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 315–324, 2015.

[10] Mor Baruch, Rafail Ostrovsky, and Will Rosenbaum. Space-time tradeoffs for distributed verification. Brief Announcement at the 35th ACM Symposium on Principles of Distributed Computing, 2016.

[11] Florent Becker, Adrian Kosowski, Martín Matamala, Nicolas Nisse, Ivan Rapaport, Karol Suchan, and Ioan Todinca. Allowing each node to communicate only once in a distributed system: shared whiteboard models. *Distributed Computing*, 28(3):189–200, 2015.

[12] Lélia Blin and Pierre Fraigniaud. Space-optimal time-efficient silent self-stabilizing constructions of constrained spanning trees. In *35th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 589–598, 2015.

[13] Lélia Blin, Pierre Fraigniaud, and Boaz Patt-Shamir. On proof-labeling schemes versus silent self-stabilizing algorithms. In *16th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 18–32, 2014.

[14] Borzoo Bonakdarpour, Pierre Fraigniaud, Sergio Rajsbaum, David Rosenblueth, and Corentin Travers. Decentralized asynchronous crash-resilient runtime verification. Technical Report CAS-16-02-BB, Department of Computing and Software, McMaster University, 2016.

[15] Keren Censor-Hillel, Eldar Fischer, Gregory Schwartzman, and Yadu Vasudev. Fast distributed algorithms for testing graph properties. *CoRR*, abs/1602.03718, 2016.

[16] Keren Censor-Hillel, Petteri Kaski, Janne H. Korhonen, Christoph Lenzen, Ami Paz, and Jukka Suomela. Algebraic methods in the congested clique. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 143–152, 2015.

[17] Atish Das-Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012.

[18] Danny Dolev, Christoph Lenzen, and Shir Peled. "tri, tri again": Finding triangles and small subgraphs in a distributed setting - (extended abstract). In *26th International Symposium on Distributed Computing (DISC)*, pages 195–209, 2012.

[19] Yuval Emek, Christoph Pfister, Jochen Seidel, and Roger Wattenhofer. Anonymous networks: randomization = 2-hop coloring. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 96–105, 2014.

[20] Laurent Feuilloley and Pierre Fraigniaud. Randomized local network computing. In *27th ACM on Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 340–349, 2015.

[21] Laurent Feuilloley, Pierre Fraigniaud, and Juho Hirvonen. A hierarchy of local decision. In *43rd International Colloquium on Automata, Languages and Programming (ICALP)*, 2016.

[22] Paola Flocchini, Guiseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Morgan & Claypool, 2012.

[23] Klaus-Tycho Förster, Thomas Luedi, Jochen Seidel, and Roger Wattenhofer. Local checkability, no strings attached. In *17th International Conference on Distributed Computing and Networking (ICDCN)*, page 21, 2016.

[24] Pierre Fraigniaud. Distributed computational complexities: are you Volvo-addicted or Nascar-obsessed? In *29th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 171–172, 2010.

[25] Pierre Fraigniaud. Locality in distributed graph algorithms. In *Encyclopedia of Algorithms*, pages 1143–1148. Springer, 2016.

[26] Pierre Fraigniaud, Cyril Gavoille, David Ilcinkas, and Andrzej Pelc. Distributed computing with advice: information sensitivity of graph coloring. *Distributed Computing*, 21(6):395–403, 2009.

[27] Pierre Fraigniaud, Mika Göös, Amos Korman, Merav Parter, and David Peleg. Randomized distributed decision. *Distributed Computing*, 27(6):419–434, 2014.

[28] Pierre Fraigniaud, Mika Göös, Amos Korman, and Jukka Suomela. What can be decided locally without identifiers? In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 157–165, 2013.

[29] Pierre Fraigniaud, Magnús M. Halldórsson, and Amos Korman. On the impact of identifiers on local decision. In *16th International Conference Principles of Distributed Systems (OPODIS)*, pages 224–238, 2012.

[30] Pierre Fraigniaud, Juho Hirvonen, and Jukka Suomela. Node labels in local decision. In *22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 31–45, 2015.

[31] Pierre Fraigniaud, David Ilcinkas, and Andrzej Pelc. Communication algorithms with advice. *J. Comput. Syst. Sci.*, 76(3-4):222–232, 2010.

[32] Pierre Fraigniaud, Amos Korman, and Emmanuelle Lebhar. Local MST computation with short advice. *Theory Comput. Syst.*, 47(4):920–933, 2010.

[33] Pierre Fraigniaud, Amos Korman, and David Peleg. Towards a complexity theory for local distributed computing. *J. ACM*, 60(5):35, 2013.

[34] Pierre Fraigniaud and Andrzej Pelc. Decidability classes for mobile agents computing. In *10th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 362–374, 2012.

[35] Pierre Fraigniaud, Sergio Rajsbaum, Matthieu Roy, and Corentin Travers. The opinion number of set-agreement. In *18th International Conference on the Principles of Distributed Systems (OPODIS)*, pages 155–170, 2014.

[36] Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. Locality and checkability in wait-free computing. *Distributed Computing*, 26(4):223–242, 2013.

[37] Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. On the number of opinions needed for fault-tolerant run-time monitoring in distributed systems. In *5th International Conference on Runtime Verification (RV)*, pages 92–107, 2014.

[38] Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. Minimizing the number of opinions for fault-tolerant distributed decision using well-quasi orderings. In *12th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 497–508, 2016.

[39] Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. Perfect failure detection with very few bits. Submitted, 2016.

[40] Pierre Fraigniaud, Ivan Rapaport, Ville Salo, and Ioan Todinca. Distributed testing of excluded subgraphs. *CoRR*, abs/1605.03719, 2016.

[41] Mika Göös and Jukka Suomela. Locally checkable proofs. In *30th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 159–168, 2011.

[42] Gene Itkis and Leonid A. Levin. Fast and lean self-stabilizing asynchronous protocols. In *35th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 226–239, 1994.

[43] Jarkko Kari, Martín Matamala, Ivan Rapaport, and Ville Salo. Solving the induced subgraph problem in the randomized multiparty simultaneous messages model. In *22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 370–384, 2015.

[44] Liah Kor, Amos Korman, and David Peleg. Tight bounds for distributed minimum-weight spanning tree verification. *Theory Comput. Syst.*, 53(2):318–340, 2013.

[45] Amos Korman and Shay Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4):253–266, 2007.

[46] Amos Korman, Shay Kutten, and Toshimitsu Masuzawa. Fast and compact self-stabilizing verification, computation, and fault detection of an MST. *Distributed Computing*, 28(4):253–295, 2015.

[47] Amos Korman, Shay Kutten, and David Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010.

[48] Moni Naor and Larry J. Stockmeyer. What can be computed locally? *SIAM J. Comput.*, 24(6):1259–1277, 1995.

[49] David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. SIAM, 2000.

[50] Fabian Reiter. Distributed graph automata. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 192–201, 2015.

[51] Stefan Schmid and Jukka Suomela. Exploiting locality in distributed SDN control. In *Proceedings of the Second ACM Workshop on Hot Topics in Software Defined Networking (HotSDN)*, pages 121–126, 2013.

The Education Column

by

Juraj Hromkovič
Department of Computer Science
ETH Zürich
Universitätstrasse 6, 8092 Zürich, Switzerland
`juraj.hromkovic@inf.ethz.ch`

Learn to Program? Program to Learn!

Matthias Hauswirth
Università della Svizzera italiana
`matthias.hauswirth@usi.ch`

Abstract

Learning to program may make students more employable, and it may make them better thinkers. However, the most important reason for learning to program may well be that it enables an entirely new way of learning.[1]

1 Why Everyone Should Learn to Program

We are in a gold rush in computer science education. Countless school districts, states, countries, non-profits, and startups *rush* to offer computer science, or coding, for all. The goal—or *gold*?—too often is seen in empowering students to get great future-proof jobs.

This first goal—*programming to earn*—is fine, but it is much too limited.

A broader goal looks at computer science education as general education that helps students to become critical thinkers. Like the headmaster of my school, who recommended I study Latin because it would make me a better thinker. It probably did. And so did studying computer science.

This second goal—*programming to think*—is great. However, I claim that there is a third, even greater, goal for teaching computer science to each and every person on the planet. Read on!

2 Computer Language as a Medium

In "Computer Science: Reflections on the Field, Reflections from the Field" [6], Gerald Jay Sussman (MIT) writes an essay called "The Legacy of Computer Science." There he cites from his own landmark programming textbook "Structure and Interpretation of Computer Programs" (SCIP) [1]:

[1] This article is based on a blog post previously published at `https://medium.com/@mathau/learning-to-program-programming-to-learn-c2c3d71d4d1d`

> The computer revolution is a revolution in the way we think and in the way we express what we think. The essence of this change is the emergence of what might best be called procedural epistemology—the study of the structure of knowledge from an imperative point of view, as opposed to the more declarative point of view taken by classical mathematical subjects. Traditional mathematics provides a framework for dealing precisely with notions of "what is." Computation provides a framework for dealing precisely with notions of "how to."

In his "Legacy of CS" essay, he then goes on about pedagogy:

> Traditionally, we try to communicate [...] skills by carefully solving selected problems on a blackboard, explaining our reasoning and organization. We hope that the students can learn by emulation, from our examples. However, the process of induction of a general plan from specific examples does not work very well, so it takes many examples and much hard work on the part of the faculty and students to transfer the skills.

And here comes the most crucial part:

> However, if I can assume that my students are literate in a computer programming language, then I can use programs to communicate ideas about how to solve problems: I can write programs that describe the general technique of solving a class of problems and give that program to the students to read. Such a program is precise and unambiguous—it can be executed by a dumb computer! In a nicely designed computer language a well-written program can be read by students, who will then have a precise description of the general method to guide their understanding. With a readable program and a few well-chosen examples it is much easier to learn the skills. Such intellectual skills are very hard to transfer without the medium of computer programming. Indeed, "a computer language is not just a way of getting a computer to perform operations but rather it is a novel formal medium for expressing ideas about methodology. Thus programs must be written for people to read, and only incidentally for machines to execute." (SCIP)

So, "computer programming" is a kind of advanced pedagogical device! Before we investigate this idea further, let's review the other two reasons for teaching programming to everyone.

3 Reason 1: Programming to Earn

The first reason mentioned in my introduction, the need for more software engineers in industry, has some benefits. There indeed seems to be a need for more professional software engineers, and software engineer is indeed a well regarded and well paid job. And learning to program in school might indeed entice some students to eventually study computer science to become professional software engineers. However, teaching programming to everyone in school, for the *sole* purpose of boosting the number of people who will later study computer science, seems like an inappropriate use of the limited time available in school.

4 Reason 2: Programming to Think

I always wondered why the headmaster of my school told me that studying Latin would teach me to think. That was, until I learned about the Trivium.

The Trivium is an idea from the Middle Ages and defines the foundational education in these times. In a presentation entitled "The Lost Tools of Learning" [9] Dorothy Sayers discussed the idea of the Trivium at Oxford University in 1947:

> The syllabus was divided into two parts: the Trivium and Quadrivium. The second part–the Quadrivium–consisted of "subjects," and need not for the moment concern us. The interesting thing for us is the composition of the Trivium, which preceded the Quadrivium and was the preliminary discipline for it. It consisted of three parts: Grammar, Dialectic, and Rhetoric, in that order.
>
> Now the first thing we notice is that two at any rate of these "subjects" are not what we should call "subjects" at all: they are only methods of dealing with subjects. Grammar, indeed, is a "subject" in the sense that it does mean definitely learning a language–at that period it meant learning Latin. But language itself is simply the medium in which thought is expressed. The whole of the Trivium was, in fact, intended to teach the pupil the proper use of the tools of learning, before he began to apply them to "subjects" at all. First, he learned a language; not just how to order a meal in a foreign language, but the structure of a language, and hence of language itself–what it was, how it was put together, and how it worked. Secondly, he learned how to use language; how to define his terms and make accurate statements; how to construct an argument and how to detect fallacies in argument. Dialectic, that is to say, embraced Logic and Disputation. Thirdly, he learned to express himself in language–how to say what he had to say elegantly and persuasively.

The Trivium taught students to think clearly, and enabled them to learn concrete subjects much more easily. Sayers argues further, that modern education (in 1947) lost track of this idea, and tried to teach students as many subjects as possible, instead of first teaching them how to think (and learn). She argues that teaching the Trivium first would make it dramatically easier and more efficient to learn any number of subjects later.

I wonder whether a modern-day equivalent of the Trivium would contain programming, or maybe, more generally "computational thinking," as one of its non-subject subjects.

In any case, if programming is part of a new Trivium—of the set of subjects that allow people to think clearly—then, obviously, everyone has to learn to program. However, in this article I want to focus on Sussman's somewhat related but different reason for learning to program.

5 Reason 3: Programming to Learn

The third reason for teaching programming to everyone is to enable Sussman's view of using computer programming as a new pedagogical device. This is a very concrete approach: if we are able to program, we can then teach other topics (like biology) by *modeling* the phenomena of those topics using computer programs.

Thus, the reason for **learning to program** is to later enable us to **program to learn**!

Note that I am not proposing that "programming to learn" would completely replace existing pedagogical tools. In fact, programming an executable model of some phenomenon may not be enough to fully understand that phenomenon. For example, if a "programming to learn" activity already starts with a rather complete specification of the phenomenon, a student may be able to implement a working program in a somewhat mechanical way, without profoundly understanding the deeper meaning of the specification or the phenomenon. Nevertheless, I believe that "programming to learn" can greatly augment existing pedagogical approaches.

I came up with the phrase "programming to learn" when reading Sussman's essay. I quickly discovered that others used it long before me. However, their interpretations differ somewhat from Sussman's idea.

5.1 Mendelsohn et al.'s Interpretation

Already in 1990, Mendelsohn et al. brought up that phrase in their book chapter on "Programming Languages in Education: The Search for an Easy Start" [4].

> [...] as new languages have been developed, there has been much discussion of whether the primary aim should be to teach children programming for its own sake, or to use programming in the service of some other end or discipline—"programming to learn, or learning to program."

Mendelsohn et al. look at the two phrases—"programming to learn" and "learning to program"—as two mutually exclusive choices, as two different ways of learning to program. They seem to regard "programming to learn" as a way to learn to program in a meaningful context, where the programming language "acts as a medium for the practicing of specific skills," while they regard "learning to program" as a way to learn to program in a decontextualized way, where the "programming language is above all a new language to be learnt."

In Sussman's view, "programming to learn" does not seem to *include* "learning to program." That is, students first learn to program, and, after that, they can use the gained programming skills to use programming to learn.

5.2 Miller's Interpretation

In his 2004 dissertation, "Promoting computer literacy through programming Python," [5] John Alexander Miller puts "programming to learn" in a more general context:

- "learning to read" enables students to "read to learn"
- "learning to write" enables students to "write to learn"
- "learning computer literacy" enables students to "use computer literacy to learn"
- "learning to program" enables students to "program to learn"

Miller's work connects "programming to think" and "programming to learn." On one hand, he considers programming as a cross-cutting part of a new Trivium, proposing to replace Latin with Python. This is more about acquiring computational thinking skills than learning to program to enable Sussman-style learning experiences. On the other hand, Miller highlights the power of programs as "executable notations" and concludes with:

> [...] integrating programming into curricular activities may significantly alter *what* knowledge becomes important to learn in many of the traditional subject areas, as well as *how* that knowledge is learned.

Sussman's idea is exactly about using programming to *alter how knowledge is learned*.

5.3 Resnick's Interpretation

In 2012, Mitchel Resnick, Professor of Learning Research at the MIT Media Lab, gave a TED talk with the title "Let's teach kids to code" [7], where he brought up the phrase "Code to Learn." He further elaborated the idea in a 2013 EdSurge post titled "Learn to Code, Code to Learn" [8]. He argues that coding helps you to learn how to: experiment with and communicate new ideas, take complex ideas and break them down into simpler parts, collaborate with other people on your projects, find and fix bugs when things go wrong, and keep persistent and persevere in the face of frustration.

Resnick's interpretation of "code to learn" seems to align with the second reason: learning to code to help students to become more literate, critical thinkers and creators. The skills he enumerates are helpful in many other contexts just like the "thinking skills" my Latin teacher instilled are helpful in other contexts. Resnick's interpretation does not, however, directly and completely address the third reason: learning to code to provide students with the means (coding) so they can learn in the entirely new way described by Sussman.

5.4 Wenger's Interpretation

In a 2012 blog post "Learning to Program, Programming to Learn" [10], Albert Wenger writes:

> Programming is "teaching" the computer how to do something. If you can't teach it to the computer you have probably not completely understood it. Hence the "programming to learn" in the subject line of this post.

He enumerates a set of school subjects in which he sees programming helping with learning: History (programming animated maps and historical timelines), English (programming word games), Music (programming music, sound, and visualizations thereof), Science (programming simulations), and Math (programmatically illustrate the number line, connect algebra and geometry).

Wenger's interpretation is very close to Sussman's. While Sussman describes the teacher writing a program for students to read, Wenger talks of the students writing the program themselves. Wenger's approach thus represents a more active approach to learning and thus is more in line with modern pedagogy. Another difference is that Sussman focused on college-level courses, while Wenger discussed using programming in school.

5.5 Guzdial's Interpretation

In his 2015 book on "Learner-centered design of computing education: research on computing for everyone" [3], Mark Guzdial, a computing education professor at the Georgia Institute of Technology, discusses a list of potential reasons for why everyone should learn computing: jobs, learn about their world, computational thinking, computational literacy, productivity, and to broaden participation.

Guzdial's book is a treasure trove of information about studies on computing education for everyone. Overall, Guzdial finds that what I call "computing to think" is less promising, because there is no evidence that one can teach general, transferable problem-solving or thinking skills by teaching programming. However, he argues and cites evidence that programming can be helpful in learning subjects like mathematics, science, or engineering:

> Writing a program to solve an engineering problem may give students new insights into the engineering problem. Writing a program to express an idea can transform how the programmer thinks about that idea. This is not about transfer. This is about the power of using a new medium, of applying computing to new domains.

This very much corresponds to Sussman's "computing to learn" perspective.

6 Programming to Learn What?

I find that idea of "programming to learn," in Sussman's sense, extremely powerful! And I realize that I and many of my colleagues have been applying that idea in our courses for a long time: we have students implement a concept as a program just so they will profoundly understand that concept.

Given that I teach concepts in computer science, the concepts I teach often are methodological and amenable to modeling and implementation in code. For example, I taught computer architecture by having students implement a simple micro-architectural simulator. Or I teach version control by having students implement a simplified simulator of Git. And who has ever taught compiler optimization without having students implement at least parts of a compiler?

I wonder where the boundaries of this "programming to learn" approach lie. Can you teach statistics this way, and how well can you do so? Biology? Economy? Psychology? Sociology? How about history, or natural languages? Carpentry? Plumbing? Or... art?

I bet the boundaries lie far beyond computer science. And I bet few non-computer-scientists actually use a "programming to learn" approach. Sometimes because they never learned to program. Other times because their students can't program.

7 Do Not Neglect Learning to Program

Programming to learn might be the most important reason for learning to program. However, in order to program to learn, we *do* have to learn to program. If we single-mindedly focus on "programming to learn," on how students can *use* programming to learn other topics, and if programming disappears—or is never introduced—as a curricular subject, then we risk that students will not learn the foundational concepts underlying "programming." As a result, they will not be able to program to learn either.

This risk is not negligible. A similar phenomenon happened in Switzerland [2], when informatics was eliminated as a subject from the high school curricula, and instead was integrated into the various subjects in which it could be applied. The result was that students did not really learn programming anymore, but only learned how to use computers as end-users.

8 Conclusions

In order to program to learn, you first have to learn to program. So we have to teach coding, or programming, to everyone! Not because they need to become professional programmers. Not even because it improves their critical thinking skills. But simply because it opens up a new, potentially powerful way of learning. By reading and writing code.

Acknowledgements

In addition to thanking the authors of the inspiring works I cited, I would like to thank all the colleagues who influenced my thinking about these ideas. Special thanks go to Judi Fusco, Patti Schank, Shuchi Grover, and Jeremy Roschelle at SRI International for the many insightful discussions about the learning sciences and computer science education. However, the interpretations and opinions presented here are my own and may not necessarily represent those of my colleagues.

References

[1] H. Abelson and G. J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, USA, 2nd edition, 1996.

[2] W. Gander. Informatics – new basic subject. In *Bulletin of EATCS*, number 116. European Association for Theoretical Computer Science, June 2015.

[3] M. Guzdial. *Learner-centered design of computing education: research on computing for everyone*. Morgan & Claypool Publishers, Nov. 2015.

[4] P. Mendelsohn, T. R. G. Green, and B. Paul. Programming languages in education: The search for an easy start. In J.-M. Hoc, T. R. G. Green, D. Gilmore, and R. Samway, editors, *Psychology of Programming*, chapter 2.5, pages 175–200. Academic Press, London, 1990.

[5] J. A. Miller. *Promoting Computer Literacy Through Programming Python*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 2004. AAI3122001.

[6] National Research Council. *Computer Science: Reflections on the Field, Reflections from the Field.* The National Academies Press, Washington, DC, 2004.

[7] M. Resnick. Let's teach kids to code. TED Talks. `http://www.ted.com/talks/mitch_resnick_let_s_teach_kids_to_code`, Nov. 2012.

[8] M. Resnick. Learn to code, code to learn. EdSurge. `https://www.edsurge.com/news/2013-05-08-learn-to-code-code-to-learn`, May 2013.

[9] D. L. Sayers. The lost tools of learning. Essay presented at Oxford University. `http://www.gbt.org/text/sayers.html`, 1947.

[10] A. Wenger. Learning to program, programming to learn. Blog post. `http://continuations.com/post/36062400780/learning-to-program-programming-to-learn`, Nov. 2012.

THE LOGIC IN COMPUTER SCIENCE COLUMN

BY

YURI GUREVICH

Microsoft Research
One Microsoft Way, Redmond WA 98052, USA
gurevich@microsoft.com

Fundamentals of p-values: Introduction

Yuri Gurevich
Microsoft Research

Vladimir Vovk
Royal Holloway

Abstract

We explain the concept of p-values presupposing only rudimentary probability theory. We also use the occasion to introduce the notion of p-function, so that p-values are values of a p-function.

The explanation is restricted to the discrete case with no outcomes of zero probability. We are going to address the general case elsewhere.

1 Prelude

Q[1]: I have a question about the Cournot Principle that one of you wrote about [3]. According to the principle, it is practically certain that a predicted event of small probability does not happen. How small should the probability be?

A: Traditionally there have been two camps of naive Cournotians, the lax ones using the 5% threshold and strict ones using the 1% threshold.

Q: Why do you call them naive?

A: A better way is to report an appropriate p-value, without committing yourself to a threshold.

Q: Oh yes, I heard about p-values and even tried to read about them but got confused. Do you have a good reference?

A: Cox and Hinkley define p-values, even though they do not use the term, in §3 of the their 1974 book "Theoretical Statistics" [2]. That is the best reference that we have.

Q: How about explaining the concept to me right now? But please take into account that, while I have been exposed to probability theory and logic, I have not studied statistics.

[1]Q and A are Quisani (a former student of the first author) and the authors respectively.

A: The general case is somewhat involved [4], but the discrete case is important and simple enough to explain the concept, especially if one presumes that every outcome has positive probability.

2 Probability trials

Recall that a discrete probability space is given by a nonempty set Ω and a function $\mathbf{P}$ from Ω to the real interval $[0, 1]$ subject to the following three requirements.

1. Ω is countable, i.e. finite or infinite countable.
2. Every $\mathbf{P}(x) \geq 0$.
3. $\sum_{x\in\Omega} \mathbf{P}(x) = 1$.

To simplify our exposition, we replace (2) with a stronger and relatively innocuous requirement

(2′) Every $\mathbf{P}(x) > 0$.

The function $\mathbf{P}$ naturally extends to subsets E of Ω: $\mathbf{P}(E) = \sum\{P(x) : x \in E\}$.

Think of the probability space $(\Omega, \mathbf{P})$ as a probability trial. Elements of Ω are outcomes, subsets of Ω are events, and $\mathbf{P}$ assigns probabilities to outcomes and events. The probability distribution $\mathbf{P}$ is the *null hypothesis* of the trial. Think of it as an alleged probability distribution. The purpose of the trial is to test $\mathbf{P}$.

Q: I guess, a predicted event of small probability, if and when it occurs, provides falsifying evidence against the null hypothesis.

A: Yes, but we do not need to define what probabilities count as small. For now, it suffices to say this: The smaller the probability of the event in question, the stronger the impugning power of the event. Suppose for example that $\Omega = \{a, b, c\}$ and that $\mathbf{P}(a) = 990/1000$, $\mathbf{P}(b) = 9/1000$ and $\mathbf{P}(c) = 1/1000$. We can order the nonempty events in the order of increasing impugning power: $\{a, b, c\}$, $\{a, b\}$, $\{a, c\}$, $\{a\}$, $\{b, c\}$, $\{b\}$, $\{c\}$.

Q: But the event $\{a, b, c\}$ has zero impugning power.

A: True, and we don't say that it has any. We just say that its impugning power is less than that of the other 6 events. Also, we ignore the empty event because it does not occur.

3 The logic angle

Q: For logicians, a discrete probabilistic trial $(\Omega, \mathbf{P})$ is a relatively simple logic structure, an extension of real arithmetic with a separate sort Outcome and a function $\mathbf{P}$ of type
Outcome $\to$ Real, subject to requirements (1)–(3) of §2.

A: In applications, a trial comes with additional information. Here are two examples, in a simplified form, from the paper you cited.

Coin Example. A coin is tossed 42 times. The null hypothesis is that all 2^{42} outcomes are equally probable. The actual outcome has 41 heads and just one tail. Clearly the impugning power of that outcome exceeds that of a random outcome.

Lottery Example. 4,000,000 people bought 10,000,000 tickets. Any of them could be the winner; thus these are 4,000,000 potential outcomes. The null hypothesis is that the probability of a person to win is proportional to the number of his or her tickets. The wife Donna of the lottery organizer John bought 3 tickets and won the lottery. Clearly the impugning power of that outcome exceeds that of a random outcome.

Q: Additional information complicates the matter. Clearly there are countless different kinds of additional information. How do statisticians deal with them?

A: They simplify the situation. They replace any additional structure with a linear preorder of the outcomes and restrict attention to the downward closed events.

Q: This is quite a simplification!

A: Often the linear preorder is given implicitly, by means of a test statistic or a nested family of events.

Q: Please remind me the necessary definitions.

4 Test tools

Let $(\Omega, \mathbf{P})$ be a probability trial. A *preorder* of a nonempty set Ω is a binary relation $\leq$ on Ω that is reflexive and transitive so that for all outcomes x, y, z we have

- $x \leq x$ and
- if $x \leq y$ and $y \leq z$ then $x \leq z$.

A preorder $\leq$ of Ω is *linear* if for all x, y we have

- $x \leq y$ or $y \leq x$.

Definition 1.

- A *test order* of Ω is a linear preorder on Ω.
- A *test pyramid* over Ω is a family of events linearly ordered by inclusion.
- A *test statistic* on Ω is a function from Ω to the reals.

Coin Example, continued. Let $\text{Heads}\,(x)$ and $\text{Tails}\,(x)$ be the numbers of the heads and tails in an outcome x respectively. The function $M(x) = \text{Min}\,\{\text{Heads}\,(x), \text{Tails}\,(x)\}$ is a test statistic. The relation $M(x) \leq M(y)$ is a linear preorder of the outcomes. Events $\{x : M(x) \leq n\}$, where $n \in \{0, 1, \ldots, 42\}$ form a test pyramid.

Given a linear preorder $\leq$ on Ω, an event $E \subseteq \Omega$ is *downward closed* if $x \leq y$ and $y \in E$ imply $x \in E$ for all x, y. The downward closed events are nested: if E_1, E_2 are downward closed then either $E_1 \subseteq E_2$ or $E_2 \subseteq E_1$. Indeed if $E_2 \nsubseteq E_1$ and $x \in E_2 - E_1$ then $E_1 \subseteq \{y : y \leq x\} \subseteq E_2$.

Note that there may be distinct outcomes x, y with $x \leq y$ and $y \leq x$; such outcomes x, y are *quasi-equal*. Linear preorders are also known as linear quasi-orders.

Test orders, test pyramids and test statistics are three tools to measure the relative strength of the impugning evidence provided by a given outcome or event. We adopt the following convention.

- For a given test order, smaller outcomes provide stronger impugning evidence.
- For a given test statistic, smaller values provide stronger impugning evidence.

And of course, for a given test pyramid of events, smaller events provide stronger impugning evidence.

In the following two sections we show that the three tools are equivalent in an appropriate sense.

5 Test orders and test pyramids

If $\le$ is a test order, let $[\le x]$ be the downward closure $\{y : y \le x\}$ of x.

Definition 2.

- Every test order $\le$ *induces* a test pyramid, namely the family of events $[\le x]$.
- Every test pyramid $\mathcal{F}$ *induces* a test order

$$\{(x, y) : (\forall E \in \mathcal{F})[y \in E \implies x \in E]\}$$

In words, the formula $(\forall E \in \mathcal{F})[y \in E \implies x \in E]$ says that every member of $\mathcal{F}$ that contains y contains x as well.

Lemma 3. *If a test order $\le$ induces a test pyramid $\mathcal{F}$ then $\mathcal{F}$ induces $\le$.*

Proof. Let $\preceq$ be the test order induced by $\mathcal{F}$.

$$\begin{aligned}
x \preceq y &\iff (\forall E \in \mathcal{F})[y \in E \implies x \in E] && \text{by the definition of } x \preceq y\\
&\iff (\forall z \in \Omega)[y \in [\le z] \implies x \in [\le z]] && \text{by the definition of } \mathcal{F}\\
&\iff x \in [\le y]\\
&\iff x \le y
\end{aligned}$$

□

Definition 4.

- Two test pyramids are *equivalent* if they induce the same test order.
- The *canonic version* $\hat{\mathcal{F}}$ of a test pyramid $\mathcal{F}$ is the test pyramid induced by the test order induced by $\mathcal{F}$.
- A test pyramid $\mathcal{F}$ is *canonic* if it is the canonic version of itself.

Q: Show me a non-canonic test pyramid.

A: Let Ω be the set of numbers $\pm\frac{1}{n}$ where $n = 1, 2, \ldots$. The precise definition of the distribution $\mathbf{P}$ on Ω is not important provided that every element of Ω has a positive probability. Let $\le$ be the standard order on the numbers $\pm\frac{1}{n}$. The desired test pyramid $\mathcal{F}$ is the collection of all subsets of Ω downward closed with respect to $\le$. Clearly $\mathcal{F}$ induces the test order $\le$. The pyramid $\hat{\mathcal{F}}$, induced by $\le$, consists of sets $[\le x]$ where $x \in \Omega$. It is a proper subcollection of $\mathcal{F}$. The difference $\mathcal{F} - \hat{\mathcal{F}}$ consists of three sets: $\emptyset$, $[\le 0]$ and Ω.

Q: You could have require that a test order $\le$ induces the pyramid of all the events downward closed with respect to $\le$. Then your counter-example would not work.

A: That is true. But then a non-canonic pyramid would be obtained from the pyramid of downward-closed events by omitting one or more of the sets $\emptyset$, $[\leq 0]$, Ω.

Q: What about canonic test orders?

A: In our current setup every test order can be viewed as canonic. We saw already that every test order is induced by a test pyramid. We'll see below that every test order is induced by a test statistic. This is not so in the general case. In fact, it is not so even in the discrete case with outcomes of zero probability.

Lemma 5. *Every pyramid $\mathcal{F}$ is equivalent to a unique canonic test pyramid, namely to $\hat{\mathcal{F}}$.*

Proof. Let $\leq$ be the test order induced by $\mathcal{F}$. By the definition, $\hat{\mathcal{F}}$ is the collection of events $[\leq x]$. Clearly $\hat{\mathcal{F}}$ induces $\leq$, and so it is equivalent to $\mathcal{F}$.

It remains to check that $\hat{\mathcal{F}}$ is the only canonic test pyramid equivalent to $\mathcal{F}$. Let $\mathcal{F}'$ be a canonic test pyramid that is equivalent to $\mathcal{F}$ and therefore induces $\leq$. Since $\mathcal{F}'$ is canonic, it is induced by $\leq$. But then $\mathcal{F}' = \hat{\mathcal{F}}$. □

Corollary 6.

1. *If a test pyramid $\mathcal{F}$ induces a test order $\leq$ then $\leq$ induces the canonic version $\hat{\mathcal{F}}$ of $\mathcal{F}$.*
2. *If a test order $\leq$ induces a test pyramid $\mathcal{F}$ then $\mathcal{F}$ is canonic.*

Proof. (1) By the definition of $\hat{\mathcal{F}}$.

(2) By Lemma 3, $\mathcal{F}$ induces $\leq$. By (1), $\mathcal{F} = \hat{\mathcal{F}}$. □

6 Test orders and test statistics

Definition 7.

- Every test order $\leq$ *induces* a test statistic $f(x) = \mathbf{P}[\leq x]$.
- Every test statistic f *induces* a test order $\{(x, y) : f(x) \leq f(y)\}$.

For a test statistics f and a real number r, let $[f \leq r]$ be the event $\{x : f(x) \leq r\}$. In the following lemma, we take advantage of having no zero-probability outcomes.

Lemma 8. *For any test order $\preceq$, we have $x \preceq y$ if and only if* $\mathbf{P}[\preceq x] \le \mathbf{P}[\preceq y]$.

Proof. If $x \preceq y$ then $[\preceq x] \subseteq [\preceq y]$ and therefore $\mathbf{P}[\preceq x] \le \mathbf{P}[\preceq y]$. This establishes the only-if implication. We prove the if implication by contrapositive. Suppose that $x \succ y$. Then x belongs to $[\preceq x]$ but not to $[\preceq y]$ so that $[\preceq x] \supsetneq [\preceq y]$. Since $\mathbf{P}(x) > 0$, we have $\mathbf{P}[\preceq x] > \mathbf{P}[\preceq y]$. ◻

Lemma 9. *For any test statistic f, we have:*

1. *If f induces a test order $\preceq$ then every* $[\preceq x] = [f \le f(x)]$.
2. $f(x) \le f(y) \iff \mathbf{P}[f \le f(x)] \le \mathbf{P}[f \le f(y)]$.

Proof.

(1) $[\preceq x] = \{y : y \preceq x\} = \{y : f(y) \le f(x)\} = [f \le f(x)]$.

(2) Follows from (1) and Lemma 8. ◻

Lemma 10. *If a test order $\preceq$ induces a test statistic f then f induces $\preceq$.*

Proof. Let $\trianglelefteq$ be the test order induced by f.

$$\begin{aligned} x \preceq y &\iff \mathbf{P}[\preceq x] \le \mathbf{P}[\preceq y] && \text{by Lemma 8,}\\ &\iff f(x) \le f(y) && \text{by the definition of } f,\\ &\iff x \trianglelefteq y && \text{by the definition of } \trianglelefteq. \end{aligned}$$ ◻

Definition 11.

- Two test statistics are *equivalent* if they induce the same test order.
- The *canonic version* of a test statistic f is the test statistic $\hat{f}(x) = \mathbf{P}[f \le f(x)]$.
- A test statistic is *canonic* if it is the canonic version of itself.

Lemma 12. *Every test statistic f is equivalent to a unique canonic test statistics, namely to $\hat{f}$.*

Proof. First we check that f is equivalent to $\hat{f}$.

$$\begin{aligned} f(x) \le f(y) &\iff \mathbf{P}[f \le f(x)] \le \mathbf{P}[f \le f(y)] && \text{by Lemma 9,}\\ &\iff \hat{f}(x) \le \hat{f}(y) && \text{by the definition of } \hat{f}. \end{aligned}$$

Second we check that $\hat{f}$ is the only canonic test statistic equivalent to f. If f' is a canonic test statistic equivalent to f then

$$\begin{aligned} f'(x) \leq f'(y) &\iff f(x) \leq f(y) && \text{because } f \text{ and } f' \text{ are equivalent,} \\ &\iff \hat{f}(x) \leq \hat{f}(y) && \text{because } f \text{ and } \hat{f} \text{ are equivalent.} \end{aligned}$$

Thus f', $\hat{f}$ and f are all equivalent and therefore induce the same order $\leq$.

$$\begin{aligned} f'(x) &= \mathbf{P}[f' \leq f'(x)] && \text{as } f' \text{ is canonic} \\ &= \mathbf{P}[\leq x] && \text{by Lemma 8, part 1} \\ &= \mathbf{P}[\hat{f} \leq \hat{f}(x)] && \text{by Lemma 8, part 1} \\ &= \hat{f}(x) && \text{as } \hat{f} \text{ is canonic.} \end{aligned}$$ □

Corollary 13.

- *If a test statistic f induces a test order $\leq$ then $\leq$ induces the canonic version $\hat{f}$ of f.*
- *The test statistic induced by any test order is canonic.*

Proof.

(1) Let f' be the test statistic induced by $\leq$. We have

$$\begin{aligned} f'(x) &= \mathbf{P}[\leq x] && \text{by the definition of } f', \\ &= \mathbf{P}[f \leq f(x)] && \text{by Lemma 9,} \\ &= \hat{f}(x) && \text{by the definition of } \hat{f}. \end{aligned}$$

(2) Let f be the test statistic induced by a given test order $\leq$. By Lemma 10, f induces $\leq$. By (1), $f = \hat{f}$. □

7 Exact p-values

In this section we define exact p-values. A more general notion of p-value will be given in the next section.

Definition 14. Given a probabilistic trial $(\Omega, \mathbf{P})$ furnished with a test order $\leq$, the *exact p-value* of an outcome x is the probability $\mathbf{P}[\leq x]$.

Notice the dependence on the test order. Of course, the test order can be given by means a test pyramid or test statistic. Test statistics are especially popular in applications. In this connection, let us give an alternative definition of exact p-values.

Definition 15. Given a probabilistic trial $(\Omega, \mathbf{P})$ furnished with a test statistic f, the *exact p-value* of an outcome x is the number $\hat{f}(x) = \mathbf{P}[f \le f(x)]$.

Of course the two definitions are consistent.

Lemma 16.

1. *If a test statistic f induces a test order $\le$ then* $\mathbf{P}[\le x] = \mathbf{P}[f \le f(x)]$.
2. *If a test order $\le$ induces a test statistic f then* $\mathbf{P}[f \le f(x)] = \mathbf{P}[\le x]$.

Proof. (1) Use Lemma 9, part 1.

(2) By Lemma 10, f induces $\le$. Now use (1). □

Q: Let's go back to the two examples of §3, compute the exact p-values of the actual outcomes and compare them.

A: The examples do not have sufficient information. We need to furnish them with reasonable test tools. For the Coin Example, we can use the test statistic $M(x) = \text{Min}\,\{\text{Heads}\,(x), \text{Tails}\,(x)\}$ defined in the continuation of the Coin Example in §4. With respect to that test statistic, the exact p-value p_1 of the actual outcome is as follows.

$$p_1 = \mathbf{P}[f \le 1] = \frac{2(1+42)}{2^{42}} < \frac{2 \times 2^6}{2^{42}} = 2^{-35}$$

Concerning the Lottery Example, one possible approach is given by the following graph G.

- The vertices of G are the 4,000,000 lottery participants plus John, the lottery organizer, whether he bought any lottery tickets or not.
- Two distinct vertices X and Y are connected by an edge in G if and only if at least one of the following conditions holds:
  - X is a parent of Y or the other way round.
  - X, Y are spouses.
  - X, Y are close friends.

Given the graph G, we have a natural test statistic $\delta(X)$ on the lottery participants, namely the length of the minimal chain of edges from John to X. If John bought at least one ticket then the minimal value of δ is 0; otherwise it is one. Donna, John's wife, is at distance 1 from John, and she bought 4 tickets. We can estimate the exact p-value p_2 of the actual outcome from below.

$$p_2 = \mathbf{P}(\delta \leq 1) \geq \frac{4}{10,000,000} > \frac{2^2}{2^4 \times 2^{10} \times 2^{10}} = 2^{-22}.$$

It follows that

$$p_1 < 2^{-35} < 2^{-22} < p_2.$$

Q: This is interesting. The win of the lottery organizer's wife seemed more striking to me than getting 41 heads in 42 coin tosses. I wonder whether p_2 is small enough to impugn the hypothesis that the lottery was fair.

A: Well, it seems safe to assume that the lottery participants at distance ≤ 1 from John bought less than 10,000 tickets. Under this assumption,

$$p_2 < \frac{10,000}{10,000,000} = \frac{1}{1000} = 0.1\%$$

which is small enough for the lax as well as strict Cournotians.

8 p-functions

In this section ε ranges over the non-negative reals.

Lemma 17. *Let f be a canonic test statistic.*

1. $\mathbf{P}[f \leq \varepsilon] = \varepsilon$ *if* $\varepsilon \in \mathit{Range}(f)$.
2. $\mathbf{P}[f \leq \varepsilon] = \varepsilon$ *if* $\varepsilon = \sup(S)$ *for some* $S \subseteq \mathit{Range}(f)$.
3. $\mathbf{P}[f \leq \varepsilon] \leq \varepsilon$ *for every* ε.

Proof. (1) If $\varepsilon = f(x)$ then

$$\begin{aligned} \mathbf{P}[f \leq \varepsilon] &= \mathbf{P}[f \leq f(x)] \\ &= f(x) && \text{as } f \text{ is canonic} \\ &= \varepsilon \end{aligned}$$

(2) If $\varepsilon = \sup(S)$ for some $S \subseteq \text{Range}(f)$ then there is a sequence $s_1 < s_2 < \dots$ of elements of S converging to ε. Then

$$\mathbf{P}[f \le \varepsilon] = \mathbf{P}[f \le s_1] + \mathbf{P}[s_1 < f \le s_2] + \mathbf{P}[s_2 < f \le s_3] + \cdots$$

so that the probability

$$\mathbf{P}[f \le s_n] = \mathbf{P}[f \le s_1] + \mathbf{P}[s_1 < f \le s_2] + \cdots + \mathbf{P}[s_{n-1} < f \le s_n]$$

converges to $\mathbf{P}[f \le \varepsilon]$ as $n \to \infty$. Therefore

$$\begin{aligned}
\mathbf{P}[f \le \varepsilon] &= \lim_{n\to\infty} \mathbf{P}[f \le s_n] \\
&= \lim_{n\to\infty} s_n && \text{by (2)} \\
&= \varepsilon && \text{by the choice of } s_1 < s_2 < \dots.
\end{aligned}$$

(3) Let $\varepsilon_0 = \sup\{s \in \text{Range}(f) : s \le \varepsilon\}$. Then $[f \le \varepsilon] = [f \le \varepsilon_0]$ and therefore $\mathbf{P}[f \le \varepsilon] = \mathbf{P}[f \le \varepsilon_0] = \varepsilon_0 \le \varepsilon$. □

Definition 18.

- A *p-function* is a test statistic f such that $\mathbf{P}[f \le \varepsilon] \le \varepsilon$ for every ε.
- A p-function f is *exact* if $\mathbf{P}[f \le \varepsilon] = \varepsilon$ for every $\varepsilon \in \text{Range}(f)$; otherwise f is *conservative*.
- Values of a p-function are *p-values*.

If f is a p-function then cf is a p-function for every $c \ge 1$. Indeed

$$\mathbf{P}[cf \le \varepsilon] = \mathbf{P}[f \le \varepsilon/c] \le \varepsilon/c \le \varepsilon.$$

If $c < 1$ then cf may not be a p-function. In particular if $\mathbf{P}[f \le \varepsilon] = \varepsilon$ for at least one ε then cf is not a p-function because, for that ε, we have

$$\mathbf{P}[cf \le c\varepsilon] = \mathbf{P}[f \le \varepsilon] = \varepsilon > c\varepsilon.$$

Theorem 19. *For any test statistic f, the following two claims are equivalent:*

1. *f is canonic.*
2. *f is an exact p-function.*

Proof. The implication (1)→(2) is proven in Lemma 17. To prove the implication (2)→(1), assume that f is an exact p-function, x is an arbitrary outcome and $\varepsilon = f(x)$. We have

$$\begin{aligned} \mathbf{P}[f \le f(x)] &= \mathbf{P}[f \le \varepsilon] \\ &= \varepsilon && \text{as } f \text{ is exact} \\ &= f(x). \end{aligned}$$

□

Corollary 20.

- *Exact p-values are values of a exact p-function.*
- *Every test order induces a unique exact p-function.*
- *Every test statistic f is equivalent to a unique exact p-function.*

9 Data snooping

Q: What about conservative p-functions? Do you really have any use for them?

A: Yes. A good example is the so-called Bonferroni correction. There is a good explanation in [1].

Q: But what is the idea?

A: Consider a version of our coin example where, for simplicity, you toss a coin only 15 times. Again the null hypothesis is that all outcomes are equally probable, and again we will use the test statistic $M(x) = \text{Min}\{\text{Heads}(x), \text{Tails}(x)\}$. Suppose that you really want to impugn the null hypothesis. So you repeat the trial over and over until you encounter an outcome where M is 1. At this point, you report that you performed a trial and got an outcome with p-value

$$\mathbf{P}[M \le 1] = \frac{2(1+15)}{2^{15}} = \frac{2^5}{2^{15}} = 2^{-10} = \frac{1}{1024}.$$

Q: But this is cheating.

A: Exactly. Yet, such data snooping occurs in science though rarely in such a blatant form; a scientist may genuinely forget about some steps in the preliminary analysis of data. In this connection, you may enjoy a relevant issue [6] of the XKCD webcomic.

Q: I remember hearing about publication bias. What is it?

A: It is a rather insidious kind of data snooping. Imagine that a large number of groups of scientists are testing an important null hypothesis. Eventually some group gets a significant p-value and submits their findings for publication while the other groups don't publish anything on the issue. Some influential statisticians argue that most published research findings are wrong and that publication bias may be the main reason for that [5].

Q: You cannot avoid multiple testing of a null hypothesis. What do you do?

A: Consider a case where the same probabilistic trial is performed n times, using test statistics $f_1, \ldots, f_n$ and getting p-values $p_1, \ldots, p_n$. Of course one would love to report the p-value $\min\{p_1, \ldots, p_n\}$ but it needs to be adjusted for the multiple testing of the null hypothesis. The Bonferroni correction is one way to adjust that p-value, by multiplying it by n. The p-value $n \cdot \min\{p_1, \ldots, p_n\}$ is valid though usually conservative. More exactly the test statistic $f = n \min(f_1, \ldots, f_n)$ is a p-function, typically conservative. Indeed,

$$\mathbf{P}[f \le \varepsilon] \le \mathbf{P}(\cup_{i=1}^{n} [f_i \le \varepsilon/n]) \le \sum_{i=1}^{n} \mathbf{P}[f_i \le \varepsilon/n] \le \sum_{i=1}^{n} \varepsilon/n = \varepsilon.$$

Acknowledgments

We thank Andreas Blass for useful comments on a draft of this paper.

References

[1] Hervé Abdi, "Bonferroni and Šidák corrections for multiple comparisons," `http://www.utdallas.edu/~herve/Abdi-Bonferroni2007-pretty.pdf`, accessed January 8, 2016. Appeared in N. J. Salkind (ed.), *Encyclopedia of Measurement and Statistics*, Sage Publications, 2007.

[2] D. R. Cox and D. V. Hinkley, "Theoretical Statistics," Chapman & Hall, 1974.

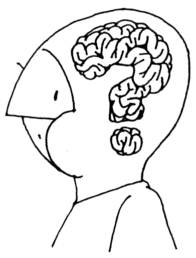
[3] Yuri Gurevich and Grant O. Passmore, "Impugning Randomness, Convincingly," Bulletin of EATCS 104, June 2011; also — with an added Prologue — in Studia Logica 82 (2012), 1–31.

[4] Yuri Gurevich and Vladimir Vovk, "Fundamentals of p-values," unpublished manuscript.

[5] John P. A. Ioannidis, "Why Most Published Research Findings Are False", PLoS Medicine, Volume 2, Issue 8, e124, August 2005.

[6] XKCD, `http://xkcd.com/882/`, accessed on Jan. 8, 2016.

News and Conference Reports



Report on BCTCS 2016

The 32nd British Colloquium for Theoretical Computer Science

22–24 March 2016, Queen's University Belfast

Amitabh Trehan

The British Colloquium for Theoretical Computer Science (BCTCS) is an annual forum in which researchers in Theoretical Computer Science can meet, present research findings, and discuss developments in the field. It also provides an environment for PhD students to gain experience in presenting their work in a wider context, and to benefit from contact with established researchers.

BCTCS 2016 was hosted by Queen's University Belfast (*QUB*), and held from 22nd to 24th March, 2016. The event attracted over 30 participants, and featured an interesting and wide-ranging programme of six invited talks and 20 contributed talks. Abstracts for all of the talks from BCTCS 2015 are provided below. We are particularly thankful to the *Heilbronn Institute for Mathematical Research* who provided bursaries for 7 PhD students (including 2 female students). The many (sponsored and non-sponsored) PhD students ensured enthusiastic participation and talks over a wide range of research areas. QUB generously provided free use of the venue, the newly opened and redesigned *Graduate School*, to promote learning among graduate students. We are also thankful to the *London Maths Society (LMS)* for their annual sponsorship of the *LMS keynote speaker in Discrete Maths* - Prof. Valerie King (University of Victoria, Canada).

The talks covered a wide range of topics in TCS and there was global participation ranging from Canada, USA, Turkey, Iceland, the UK, of course, and the republic of Ireland. The opening talk was given by Matthew Hennessy (Trinity College Dublin). Matthew gave an overview of recent research in transactional distributed systems outlining semantic theories for process calculus for co-operating transactions. Michael Butler (Southampton) continued the theme in the afternoon with a talk on Event Refinement Structures (ERS) for the Event-B formal method.

Wednesday began with an interesting talk by Magnus M Halldórsson (Reykjavik University) on the philosophical and practical questions involving choosing the right problems to work on continuing with his recent results involving scheduling problems arising from wireless networks. The morning continued with Gregory Chockler (Royal Holloway) talking about using erasure codes with replication for reliable storage in asynchronous distributed systems. In the pre-lunch session, Bhaskar DasGupta (University of Illinois, Chicago, USA) gave a talk featuring non-trivial bounds on node expansions and cut-sizes for Gromov-hyperbolic graphs (or, hyperbolic graphs for short).

The session continued with a talk from our LMS keynote speaker, Valerie King. Valerie is an ACM fellow who is well known for her work on algorithms (in particular, dynamic and distributed algorithms). Valerie with Jared Saia (University of New Mexico) and others have made extensive and fundamental advances in solving the very important problem of Byzantine agreement. They have used a number of innovative techniques including spectral methods for achieving their results. In her talk, she described work on efficient algorithms for Byzantine agreement without secrecy by making connections with a new collective coin flipping problem. The afternoon also featured a stroll to the nearby Botanic gardens and the Ulster museum.

The final day began with another interesting talk- by Bruce Kapron (University of Victoria, Canada) on Gambling, information and encryption security. The talk featured many animated race horses, Yao's question (1982)- can encryption security may be characterized using computational information? His talk would have probably benefited many bookies before the Grand National. Cassio D. Campos (QUB) gave a talk on Inferential Complexity in Probabilistic Graphical Models which should probably have featured on day one considering the excellent introduction to Belfast that it gave! The final invited speaker was Robert Giles (Economics, QUB) who gave a talk on the underlying game theoretic concepts behind consent in network formation, an area in which he has extensive experience.

BCTCS 2017 will be hosted by St Andrews University, Researchers and PhD students wishing to contribute talks concerning any aspect of Theoretical Computer Science are cordially invited to do so. Further details are available from the BCTCS website at `www.bctcs.ac.uk`.

Invited Talks at BCTCS 2016

Michael J Butler (Southampton University)
Verification Patterns for Refinement

Event-B is a general purpose refinement-oriented formal method. Event Refinement Structures (ERS) provide additional structure for refining Event-B models, in particular, for refining course-grained atomicity to fine-grained atomicity when reasoning about concurrent and distributed systems. This talk presents specification-oriented patterns for ERS refinement and verification. A specification-oriented pattern is determined by the shape of the problem description rather than the solution description.

Robert Giles (Queen's University Belfast)
Consent in Network Formation: Game Theoretic Solutions

We consider the formation of networks under the principle of mutual consent and costly link formation. This problem has attracted considerable game theoretic analysis that we survey in this presentation. First, we consider link-based stability concepts founded on the seminal work by Jackson and Wolinsky (1996). In particular, we survey some refinements of pairwise stability notions in the context of costly link formation. Second, we look at non-cooperative game theoretic analysis of consent in link formation known as the Myerson network formation game. The deficiency of the Nash equilibrium concept is shown. Instead a belief-based equilibrium concept, known as a self-confirming equilibrium, is explored to model meaningful network formation in this context. An equivalence with strictly pairwise stable networks is shown. Finally, we explore the role of correlations of payoff functions through the notion of a potential. If network payoffs admit a potential function, dynamic network formation algorithms can be devised that converge to strictly pairwise stable networks that are supported as a self-confirming monadic equilibrium in the standard Myerson network formation game.

Magnus M Hallsdórsson (Rejkjavik University)
"What problem should I solve?" and Efficiency in Wireless Networks

The selection of topic to work on is perhaps the most important aspect of research, and one fraught with pitfalls. When we determine significance of a problem from the "importance in applications", it is important to understand the assumptions underlying the formulations, since all abstractions leak somewhere. We ponder these issues while examining recent progress in scheduling problems underlying wireless networking. The modeling of reality, in this case "interference", is crucial. Simple abstractions can be valuable, as long as we are aware of their limitations. This brings up the issue of wider interest: how well can simpler abstractions approximate more detailed/complex ones?

Matthew Hennessy (Trinity College Dublin)
Behavioural Theories for Co-operating Transactions

Relaxing the isolation requirements on transactions leads to systems in which transactions can co-operate to achieve distributed goals. However in the absence of isolation it is not easy to understand the desired behaviour of transactional systems, or the extent to which the other standard ACID properties of transactions can be maintained: atomicity, consistency and durability. In this talk I give an overview of some recent research in this area, outlining semantic theories for a process calculus augmented by a new construct for co-operating transactions. In particular I focus on property logics which can be used to distinguish behaviourally between such transactions.

Bruce Kapron (University of Victoria, Canada)
Gambling, Computational Information and Encryption Security

We revisit the question, originally posed by Yao (1982), of whether encryption security may be characterized using computational information. Yao provided an affirmative answer, using a compression-based notion of computational information to give a characterization equivalent to the standard computational notion of semantic security. We give two other equivalent characterizations. The first uses a computational formulation of Kelly's (1957) model for "gambling with inside information", leading to an encryption notion which is similar to Yao's but where encrypted data is used by an adversary to place bets maximizing the rate of growth of his wealth over a sequence of independent, identically distributed events. The difficulty of this gambling task is closely related to Vadhan and Zheng's (2011) notion of KL-hardness, a form of which is equivalent to a conditional form of the pseudoentropy introduced by Håstad et. al. (1999). Using techniques introduced to prove this equivalence, we also give a characterization of encryption security in terms of conditional pseudoentropy. Finally, we reconsider the gambling model with respect to adversaries with linear utility in an attempt to understand whether assumptions about the rationality of adversaries may impact the level of security achieved by an encryption scheme. (Joint work with Mohammad Hajiabadi.)

Valerie King (University of Victoria, Canada)
Tossing a Collective Coin and Coming to Agreement

Over 35 years ago, Leslie Lamport formulated a fundamental problem of coordination in a distributed network. He asked us to imagine an army led by generals, who send messages to each other with the goal of coming to agreement on a strategy. Planted among those generals are spies who seek to thwart this goal. Not long after this Byzantine agreement problem was presented, there were a few developments: an impossibility for any deterministic scheme, a randomized exponential time algorithm, and a demonstration that one globally known coin toss could solve the problem in constant expected time.

Some researchers turned to the use of committed secret coinflips via cryptography, while others turned to the study of collective coin flipping with full information. Recently, the need for Byzantine agreement without the overhead of cryptographic techniques has arisen in decentralized digital currency systems.

I will describe joint work with Jared Saia and others on efficient algorithms for Byzantine agreement without secrecy, including the first polynomial time algorithm for this problem in a fully asynchronous model, which is obtained by solving a new collective coin flipping problem.

Contributed Talks at BCTCS 2016

Athraa Al-Krizi (University of Liverpool)
Probabilistic Model Checking of One-Dimensional Nano Communication System

Molecular communication is a bio-inspired paradigm in which molecules are transmitted, propagated and received between nanoscale machines. Establishing controlled molecular transmissions between theses nanomachines represents a major challenge. Many studies have aimed to model the physical medium (channel) of molecular communication, primarily from a communication or information-theoretical perspective.

In this talk we model a simple time-slotted communication system between nanoscale machines in a one-dimensional environment. This communication system employs some bio-inspired rules that can be checked at each interval. The system model has been verified using the probabilistic model checking tool PRISM on different sized networks. We were able to verify that acknowledgement has been obtained, and thus, communication between these nanonodes has been ascertained. The results were promising for further study of more complex scenarios such as multi-access channels.

Thomas van Binsbergen (Royal Holloway, University of London)
Executable Component-Based Semantics

To improve the practicality of formal semantic definitions, the PLanCompS project has developed a component-based approach. In this approach, the semantics of a language is defined by translating its constructs to combinations of so-called fundamental constructs, or 'funcons'. Each funcon is defined using a modular variant of structural operational semantics, and forms a language-independent component that can be reused in definitions of different languages.

For specifying component-based semantics, we have designed and implemented a meta-language called CBS. CBS includes specification of abstract syntax, of its translation to funcons, and of the funcons themselves. In this talk we discuss the compilation of CBS funcon specifications to Haskell code. In particular, we shall discuss how modularity is obtained in Haskell definitions of funcons.

Joshua Blinkhorn (University of Leeds)
Dependency Schemes: Semantics and Soundness in QBF Calculi

The tremendous success of SAT solvers in recent years has lead to a natural extension to quantified Boolean formula (QBF) solving. Whereas SAT is the canonical NP-complete decision problem, deciding QBF is PSPACE-complete, and captures the problem of determining winning strategies in two-player games with perfect information. This semantic interpretation is central to the recent methods for proving lower bounds via the strategy extraction paradigm.

The linear ordering of a QBF's quantifier prefix imposes a trivial "dependency structure" upon its variables which, unfortunately, identifies dependencies which are not essential for particular instances. A dependency scheme is an algorithm which attempts to identify such spurious dependencies directly from the syntactic form of an instance; the results are used in state-of-the-art QBF solving to optimise performance. Since every unsuccessful run of a solver provides a proof of falsity, this raises questions about the underlying dependency proof systems: What is their proof complexity relative to other QBF calculi? For which dependency schemes are the underlying proof systems sound? Is strategy extraction possible? The suggestion also arises to implement dependency schemes in stronger calculi, for example in the QBF analogue of propositional Frege systems. Employing a semantic framework, the talk will present some new results, targeting an improved understanding of variable dependency in QBF calculi.

Michele Bottone (University of Middlesex)
The Agoric Process

Many phenomena can be viewed as systems arising from the interactions of agents, where an important aspect is computation, defined as the abstract representation of a process in terms of states and transitions. Such computational systems have the ability to share information and allocate resources and to parcel the computation in efficient ways through some form of signalling; they also occur in dynamically changing environments with asynchronous and unpredictable changes, including the possibility of new agents entering the system or leaving it. This basic infrastructure of open systems shares many similarities with a marketplace, where a collection of agents meet to exchange things of value to its participants. We use the term "agoric processes" to capture the mathematical essence of such information-processing mechanisms.

In this talk, I argue that the rise of connected and autonomous systems provides a useful backdrop both for experimentation and attaining a mathematical theory of distributed computation, and present a formalisation of the intuitive notion of an agoric process as a computational object living on some graph structure together with information flows. In this setting, information is an equivalence class of all ways of describing the same information possessed by agents, and one can use the Rota-Wallstrom theory of integration of functions indexed by set partitions to represent higher-level concepts.

Cassio P De Campos (Queen's University Belfast)
Inferential Complexity in Probabilistic Graphical Models

Computations such as evaluating posterior probability distributions and finding joint value assignments with maximum posterior probability are of great importance in practical applications of probabilistic graphical models. These computations, however, are intractable in general, both when the results are computed exactly and when they are approximated. In order to successfully apply probabilistic graphical models in practical situations, it is crucial to understand what does and what does not make such computations hard. In this talk we guide the audience through some of the most important computational complexity proofs and give insights about the boundary between tractable and intractable.

Gregory Chockler (Royal Holloway, University of London)
Space Bounds for Reliable Storage: Fundamental Limits of Coding

We study the space requirements of reliable storage algorithms in asynchronous distributed systems. A series of recent works have advocated using coding-based techniques (and in particular, erasure codes) as a way of reducing space overheads incurred by the standard replication approaches. However, a closer look reveals that they incur extra costs in certain scenarios. Specifically, if multiple clients access the storage concurrently, then existing asynchronous code-based algorithms may store a number of copies of the data that grows linearly with the number of concurrent clients. We establish a lower bound showing that this limitation is indeed inherent. We also present a reliable storage algorithm with matching space complexity thus proving that our bound is tight. Our algorithm is based on a new technique combining erasure codes with replication so as to obtain the best of both. I will start by introducing and motivating the problem, followed by an overview of the key ideas and techniques behind our results. (Joint work with Yuval Cassuto, Idit Keidar and Alexander Spiegelman.)

Bhaskar DasGupta (University of Illinois, USA)
Node Expansions and Cuts in Gromov-hyperbolic Graphs

Gromov-hyperbolic graphs (or, hyperbolic graphs for short) represent an interesting class of "non-expander" graphs. Originally conceived by Gromov in 1987 in a different context while studying fundamental groups of a Riemann surface, the hyperbolicity measure for graphs has recently been a quite popular measure in the network science community in quantifying "curvature" and "closeness to a tree topology" for a given network, and many real-world networks have been empirically observed to be hyperbolic.

In this talk, we give constructive non-trivial bounds on node expansions and cut-sizes for hyperbolic graphs, and show that witnesses for such non-expansion or cut-size can in fact be computed in polynomial time. We also provide some algorithmic consequences of these bounds and their related proof techniques for a few problems related to cuts and paths for hyperbolic graphs, such as the existence of a large family of s-t cuts with small number of cut-edges when s and t are at least logarithmically far apart, efficient approximation of hitting sets of size-constrained cuts, and a polynomial-time solution for a type of small-set expansion problem originally proposed by Arora, Barak and Steurer.

Colm Ó Dúnlaing (Trinity College Dublin)
An almost-confluent congruential language which is not Church-Rosser con-

gruential

It is fairly easy to show that every regular set is an almost-confluent congruential language (ACCL), and Diekert et al (2015) showed that every regular set is a Church-Rosser congruential language (CRCL). The existence of an ACCL which is not a CRCL remained an open question. In this talk we present one such ACCL.

Marie Farrel (Maynooth University, Ireland)
A Logical Framework for Integrating Software Models via Refinement

Modern software development focuses on model-driven engineering: the construction, maintenance and integration of software models, ranging from formal design documents through to program code. We frequently model software at different levels of abstraction, starting with a very high level abstract specification and finishing with a detailed concrete implementation. In formal software engineering we can map between these levels of abstraction in a verifiable way through a process known as refinement. The question is how to combine information from models which focus on different aspects of the software system.

The theory of institutions observes that once the syntax and semantics of a formal system have been defined in a uniform way, a set of specification building operators can be defined that allow you to write, modularise and build up specifications that can be defined in a formalism-independent manner. Event-B is a formal specification language that enables the user to prove safety properties of a specification. It facilitates the modelling of systems at different levels of abstraction through the verifiable process of refinement. Our goal is to represent the Event-B formalism in terms of institutions and provide modularisation constructs which increase the scalability of Event-B for use in larger projects. A benefit of this approach is the increased interoperabilty of Event-B via institution comorphisms to allow aspects of the system to be specified in different formalisms and included in the final Event-B specification.

Andrew Healy (Maynooth University, Ireland)
Evaluating SMT solvers for software verification

SMT (Satisfiability Modulo Theories) solvers are an important component in deductive software verification systems. Such systems usually differ greatly in terms of specification language and approach to intermediate verification condition generation. This variety hinders the development of a common benchmark suite and makes the comparative evaluation of verification systems difficult. By using a large benchmark suite designed to test the capabilities of SMT solvers beyond software verification (in problem domains such as operations research and cryptology, for example), we aim to identify a subset of the large benchmark suite that can be used as a surrogate suite for software verification projects. We use dynamic profiling to obtain a feature vector that characterises the workload of the solver before using techniques from cluster analysis to form new suites based on the observed behaviour of the solver under a verification workload.

We present the workflow and results of this process. A useful outcome will be a prediction of the most appropriate solver or combination of solvers to choose for a given verification problem. Such a model will show the correspondence of input to result in a way that would be useful to verification system and SMT tool developers as well as end users.

Ruth Hoffman (University of Glasgow)
Autonomous Agent Behaviour Modelled In PRISM

With the rising popularity of autonomous systems and their increased deployment within the public domain, ensuring the safety of these systems is crucial. Although testing is a necessary part in the process of deploying such systems, simulation and formal verification are key tools, especially at the early stages of design. Simulation allows us to view the continuous dynamics and monitor the behaviour of a system. On the other hand, formal verification of autonomous systems allows for a cheap, fast, and extensive way to check for safety and correct functionality of autonomous systems that is not possible using simulations alone. In this talk I demonstrate a simulation and the corresponding probabilistic model of an unmanned aerial vehicle (UAV) in an exemplary autonomous scenario and present results of both models. Further, I propose a definition of autonomy which can be used to model autonomous systems, followed by a discussion on how simulations inform probabilistic models.

Alison Jones (Swansea University)
Extracting Monadic Parsers from Proofs

My talk outlines a proof-theoretic approach to developing correct and terminating monadic parsers. Using modified realizability, we extract formally verified and terminating programs from formal proofs. By extracting both primitive parsers and parser combinators, it is ensured that all complex parsers built from these are also correct, complete and terminating for any input. We demonstrate the viability of our approach by means of two case studies: we extract (1) a small arithmetic calculator and (2) a non-deterministic natural language parser. The work is being carried out in the interactive proof system Minlog. (Joint work with Ulrich Berger and Monika Seisenberger)

Josh Lockhart (University College London)
An entanglement detection problem in a faulty quantum computer.

Quantum computers and algorithms promise to be a very disruptive technology. A key ingredient in algorithms that run on a quantum device is the ability for the hardware components to be entangled with one another. If two quantum objects become entangled, then the simple act of checking the state of one object can instantaneously affect the state of the other. A great deal of effort has been expended studying the questions of what quantum entanglement really is, how to create it, and how to keep it around long enough to perform useful work. In this talk we outline new combinatorial techniques for verifying the existence of entanglement in a quantum computer. We show how graph theory can be used to think about a particular model of a faulty quantum computer, in an attempt to gain complexity theoretic insight into the problem of checking for the existence of entanglement. Specifically, we consider a restricted class of quantum states that can be represented by the combinatorial Laplacian matrix of a graph. This object encodes the states of a quantum computer that has suffered from a particular kind of error in its operation: the computer promised to construct a certain state but instead it has erroneously constructed one of a number of alternative states. We prove that our graph representation allows for certain well known entanglement criteria to be re-expressed in terms of the structure of the graph corresponding to the quantum state. Hence, the entanglement, or lack theoreof, in the quantum computer after the fault can be tested via this purely combinatorial method.

David Kohan Marzagao (King's College London)
Flag Coordination Games and Random Walks

The main goal of this talk is to establish and explore a connection between flag coordination games and random walks. A Flag Coordination Game can be seen as graph colouring game where, in each round, each node colours itself based on an algorithm and on the range of visibility this node has. For example, assume you have 20 people in a circle playing a flag coordination game, each of them holding a red flag and a blue flag. Their goal is to reach a configuration where each of them is raising a different flag from both neighbours, i.e., to achieve a proper colouring of the graph. Their visibility is limited in that they can only see the flags raised by their immediate neighbours. The game starts with random flags and players follow an algorithm to decide which flag to raise at each new round. More precisely, if their neigbours are raising the same colour, chose the other colour, else, randomize between red and blue. What is the probability that they eventually reach their goal? What is the expected number of rounds for the game to finish? By proving an equivalence between such a game and a game of annihilating random walking particles, we can prove a theorem regarding the probabilities involved in the game, as well as an upper bound for the expected time it takes for the game to end. We can, then, generalize some of the results for graphs such that every vertex has degree 2.

Brett McLean (University College London)

The Finite Representation Property for Composition, Intersection, Domain and Range.

Motivated by modelling collections of deterministic programs, one might be interested in algebras isomorphic to a set of partial functions equipped with some set-theoretically-defined operations. We call such algebras representable and call the isomorphisms representations. The finite representation property holds for a signature of operations if any finite representable algebra can be represented using only a finite set as a base for the partial functions. I will describe a proof that the finite representation property holds for some of the most expressive signatures considered, containing the composition, intersection, domain and range operations. (Joint work with Szabolcs Mikulás.)

Reino Niskanen (University of Liverpool)

Undecidability of 2-dimensional Robot Games and Other Attacker-Defender Games

In this talk we consider simple two-player vector additional games, called robot games. In robot games, two players, Adam and Eve, are given sets of moves which they use to push a token on the integer lattice $\mathbb{Z}^n$ starting from some initial point; Eve tries to push the token to the origin, whilst Adam tries to prevent this. The decision problem of the game is to determine which of the players has a strategy that guarantees victory. By constructing a game that simulates a 2-counter Minsky machine, we show that it is undecidable whether Eve has a winning strategy already when the game is played on a plane $\mathbb{Z}^2$.

We also consider other two-player games, called Attacker-Defender games, that generalize robot games by having more complex state spaces and sets of moves, allowing or disallowing certain moves depending on the internal states of players. We consider games played on topological braids where Eve tries to unbraid a braid, or on vectors, where the moves are linear transformations rather than addition of vectors in robot games. We show that it is also undecidable whether Eve has a winning strategy in these games by constructing games that simulate one-counter automata on infinite words. (Joint work with Vesa Halava, Tero Harju, Igor Potapov and Julien Reichert).

Daniel Playfair (Queen's University Belfast)

Selection of optimal checkpoints from query plan graphs

Database fault tolerance is important for supporting critical business operations. The existing approach is to provide replicated fault tolerant data stores. Such solutions protect data and can offer availability. However, in the event of failures, work being performed at the time of execution is lost. Existing research into solving this problem and providing intra-query fault tolerance focuses on distributed or row-oriented databases. Such solutions are not suitable for use with the column-oriented in-memory databases increasingly used for high-performance workloads.

A key requirement to providing intra-query fault tolerance is the ability to devise an efficient checkpoint plan for a given query plan. We introduce and discuss a general model for reasoning about query plans. We make observations about the location of worst case queries in such plans, and introduce algorithms for calculating the worst case execution time of checkpointed queries. We also discuss aspects of algorithms for producing checkpoint plans within such a model.

Craig Reilly (University of Glasgow)
Enumeration of knots using constraint programming

This presentation details a novel approach, using constraint programming, to the problem of enumerating knot diagrams. Our enumeration makes use of Gauss code representations of knot diagrams, which leads to an obvious modelling as a constraint satisfaction problem. However, a Gauss code may not always represent a knot diagram (they might represent a virtual knot) and we show that we can add constraints to our model to disallow virtual knots. Further, each knot diagram can be represented by many Gauss codes, and we explain new ways of modelling the enumeration with a view towards symmetry breaking.

Latif Salum (Dokuz Eylul University, Turkey)
The Tractability of Un/Satisfiability

A safe acyclic Petri net (PN) is associated with some Exactly-1 3SAT formula $\phi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$, in which a clause $c_k = (z_i \dot{\vee} z_j \dot{\vee} z_u)$ is an exactly-1 disjunction $\dot{\vee}$, rather than disjunction $\vee$, of three literals: c_k is true exactly when only one of z_i or z_j or z_u is true. Some 2SAT/XOR-SAT formula arising in the *inversed* PN checks if the truth assignment of a literal (a transition firing) z_v is "incompatible" for the satisfiability of the 3SAT formula (the reachability of the target state in the *inversed* PN). If z_v is incompatible, then z_v is discarded and $\bar{z}_v$ becomes true. Therefore, a clause $(\bar{z}_v \dot{\vee} z_i \dot{\vee} z_j)$ *reduces* to the conjunction $(\bar{z}_v \wedge \bar{z}_i \wedge \bar{z}_j)$, and a 3-literal clause $(z_v \dot{\vee} z_u \dot{\vee} z_x)$ *reduces* to the 2-literal clause $(z_u \oplus z_x)$. This reduction facilitates checking (un)satisfiability; the 3SAT formula is (un)satisfiable iff the target state of the *inversed* PN is (un)reachable. The solution complexity is $O(n^5)$. Therefore, it is the case that $\mathcal{P} = \mathcal{NP} = \text{co-}\mathcal{NP}$.

Chhaya Trehan (Queen's University Belfast)
Energy consumption of parallel workloads: convexity, parallelism and memory intensity

Performance and energy consumption are important and contradicting design criteria for modern multicore processors. Optimizing one whilst imposing a threshold on the other leads to two flavors of optimization: in the laptop problem, the goal is to maximize performance given a fixed energy budget; in the server problem, the goal is to minimize energy consumption given a fixed performance budget. We deal with the server problem in this talk. Energy and performance of a parallel application running on a multicore chip are convex functions of its varying operating frequency. Analytical frameworks have been built that exploit this in order to tune the operating frequency of the processor. However, existing theoretical work on energy minimization using frequency tuning ignores the time and energy consumed by the processor whilst it waits to access the data it requires from the main memory. We present a new energy-performance model which accounts for the time and energy consumed by a processor on memory accesses in addition to the time and energy consumed on actual CPU instructions. We show that the problem of energy minimization under a performance budget remains a convex optimization problem in the new model. We also investigate how the optimal operating points (frequencies) in our model differ from the operating points in the model that does not account for the energy performance overhead of memory accesses. Finally, we show the relationship between the optimal frequencies and energy-aware scheduling of an application.

Report from the Japanese Chapter

Ryuhei Uehara (JAIST)

EATCS-JP/LA Workshop on TCS and Presentation Awards

The 13th *EATCS/LA Workshop on Theoretical Computer Science* was held at Research Institute of Mathematical Sciences, Kyoto University, January 26 to January 28, 2015. (The program also can be found at `http://www.is.titech.ac.jp/~mori/LA2015/LA2015winter_program.pdf`, but it is written in Japanese...)

By attendees' voting, the following talk by **Prof. Yukiko Yamauchi** (Kyushu University) was selected as the 13th EATCS/LA Presentation Award:

> *Pattern Formation by Oblivious Synchronous Mobile Robots in the Three Dimensional Space* by Yukiko Yamauchi (Kyushu University), Taichi Uehara (Kyushu University), and Masafumi Yamashita (Kyushu University)

The award will be given to her at the Summer LA Symposium held in July 2016.

We also established another presentation award, named "EATCS/LA Student Presentation Award" to encourage students. **Mr. Shuichi Hirahara** (The University of Tokyo) who presented the following paper, was selected as the fifth EATCS/LA Student Presentation Award:

> *Limits of Minimum Circuit Size Problem as Oracle* by Shuichi Hirahara (The University of Tokyo) and Osamu Watanabe (Tokyo Institute of Technology)

The award (with some gift for playing in his laboratory) has been already recognized publicly at the last day, January 28, 2015.

Congratulations!

This workshop is jointly organized with *LA symposium*, Japanese association of theoretical computer scientists. Its purpose is to give a place for discussing topics on all aspects of theoretical computer science. (In fact, I've heard some different opinions that "L" stands for Logic and/or Language, and "A" stands for Algorithm and/or Automaton.) That is, this workshop is an unrefereed familiar meeting. All submissions are accepted for the presentation. There should be no problem of presenting these papers in refereed conferences and/or journals. We hold it twice a year (January/February, and July/August). If you have a chance, I recommend you to attend it. Check `http://www.ecei.tohoku.ac.jp/alg/EATCS-J/` for further details. You can find the program of the last workshop below.

Program of EATCS-JP/LA workshop on TCS (January 26th to 28th, 2016)

In the following program, "*" indicates ordinary speakers, while "**" indicates student speakers. The number [S*xx*] means student session, which consists of shorter talks than the ordinary talks.

[S1] SUS queries on run-length-encoded string
***Takuya Mieno, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda (Kyushu University)*

[S2] Data structure and algorithm for longest common extension queries
***Yuka Tanimura (Kyushu University), Tomohiro I (Kyushu Institute of Technology), Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda (Kyushu University)*

[S3] Approximate tree distance for a large number of data
***Takumi Yone, Yukiko Yamauchi, Shuji Kijima, Masafumi Yamashita (Kyushu University)*

[S4] Regret Analysis for Online Binary Search Tree Problems with Switching Costs
***Tadahiro Matsukawa, Yukiko Yamauchi, Shuji Kijima, Masafumi Yamashita (Kyushu University)*

[1] Pattern Formation by Oblivious Synchronous Mobile Robots in the Three Dimensional Space
**Yukiko Yamauchi (Kyushu University), Taichi Uehara, Masafumi Yamashita (Kyushu University)*

[2] An efficient data structure for alignments of substrings
**Yoshifumi Sakai (Graduate School of Agricultural Science, Tohoku University)*

[3] Folding Orthogonal Polygons to Orthogonal Boxes
*Takashi Horiyama, **Koichi Mizunashi (Saitama University)*

[S5] Online prediction for non-cumulative loss functions
***Wakana Mori, Kohei Hatano, Eiji Takimoto (Kyushu University)*

[S6] Theory and Practice of Dynamic Graph Algorithms
***Mikiya Imura (Tokyo Institute of Technology)*

[S7] On repetition factorization of strings.
***Hiroe Inoue, Shyunsuke Inenaga , Hideo Bannai (Kyushu University), Masayuki Takeda (kyushu University)*

[S8] Inferring Strings from Lyndon Tree
***Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda (Kyushu University)*

[4] Lower bounds of circuit-size loss for direct-product lemmas
**Akinori Kawachi (Tokushima University)*

[5] On the Computational Complexity of Counting Gaps in Numerical Semigroups
**Shunichi Matsubara (Aoyama Gakuin University)*

[6] Limits of Minimum Circuit Size Problem as Oracle
***Shuichi Hirahara (The University of Tokyo), Osamu Watanabe (Tokyo Institute of Technology)*

[7] Key Dependent Message Security in the Random Oracle Model

***Fuyuki Kitagawa (Tokyo Institute of Technology, AIST), Takahiro Matsuda, Goichiro Hanaoka (AIST), Keisuke Tanaka (Tokyo Institute of Technology, JST CREST)*

[8] Selfless Anonymity on Group Signature

***Ai Ishida (Tokyo Institute of Technology / AIST), Keita Emura (NICT), Goichiro Hanaoka, Yusuke Sakai (AIST), Keisuke Tanaka (Tokyo Institute of Technology / JST CREST), Shota Yamada (AIST)*

[9] On Efficient Correctability of Samplable Errors

**Kenji Yasunaga (Kanazawa University)*

[S9] Analysis of Black Box Reduction for Circuits

***Ryo Ashida (Tokyo Institute of Technology)*

[S10] Attribute-Based Encryption from Lattices with Revocation

***Yuuki Sawai, Yuyu Wang (Tokyo Institute of Technology), Keisuke Tanaka (Tokyo Institute of Technology/CREST)*

[S11] A secure two-party protocol using indistinguishability obfuscators

***Kanako Baba (Tokushima University), Akinori Kawachi*

[S12] Sliding tokens on unicyclic graphs

***Duc Anh Hoang (JAIST), Ryuhei Uehara (JAIST))*

[S13] On Left-Right Maximal Generic Words

***Yuta Fujishige, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda (Kyushu University)*

[S14] On abelian square-free strings

***Takafumi Inoue, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda (Kyushu University)*

[10] A revisit of tangles from perspectives of ideals and filters

**Koichi Yamazaki (Gunma University)*

[11] Approximation of ASPL on graph of diameter 3

***Nobutaka Shimizu, Ryuhei Mori (Tokyo Institute of Technology)*

[12] What Is a Network Community? A Novel Quality Function and Detection Algorithms

***Atsushi Miyauchi, Yasushi Kawase (Tokyo Institute of Technology)*

[13] Efficiency of Garbled Circuits for Symmetric Functions

***Hiroyuki Tohyama, Zen Inomata (Tokyo Institute of Technology), Keisuke Tanaka (Tokyo Institute of Technology, JST CREST)*

[14] A Polynomial-time Algorithm for Checking the Inclusion of Deterministic Restricted One-Counter Transducers Which Accept by Final State

**Mitsuo Wakatsuki, Etsuji Tomita, Tetsuro Nishino (The University of Electro-Communications)*

[15] A column generation approach for the bus crew scheduling problem

***Yuki Sawai, Yannan Hu, Wei Wu (Nagoya University), Hideki Hashimoto (Tokyo University of Marine Science and Technology), Masaki Kato, Tsutomu Saito (Kozo Keikaku Engineering Inc.), Mutsunori Yagiura (Nagoya University)*

[S15] Randomized Approximate Counting of the Number of Paths in a Grid Graph

***Yuki Shibata, Yukiko Yamauchi, Shuji Kijima, Masafumi Yamashita (Kyushu University)*

[S16] Efficient Enumeration of Series-Parallel Graphs

***Atsushi Fujii (JAIST), Ryuhei Uehara (JAIST)*

[S16] An algorithm for computing minimal absent words

***Yuki Tsujimaru, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda (Kyushu University)*

The Japanese Chapter

Miscellaneous

EATCS Fellows' Advice to the Young Theoretical Computer Scientist

Luca Aceto (Reykjavik University)
with contributions by Mariangiola Dezani-Ciancaglini,
Yuri Gurevich, David Harel, Monika Henzinger,
Giuseppe F. Italiano, Scott Smolka,
Paul G. Spirakis and Wolfgang Thomas

I have always enjoyed reading articles, interviews, blog posts and books in which top-class scientists share their experience with, and provide advice to, young researchers. In fact, despite not being young any more, alas, I feel that I invariably learn something new by reading those pieces, which, at the very least, remind me of the things that I *should* be doing, and that perhaps I am *not* doing, to uphold high standards in my job.

Based on my partiality for scientific advice and stories, it is not overly surprising that I was struck by the thought that it would be interesting to ask the EATCS Fellows for

- the advice they would give to a student interested in theoretical computer science (TCS),
- the advice they would give to a young researcher in TCS and
- a short description of a research topic that excites them at this moment in time (and possibly why).

In this article, whose title is inspired by the classic book *Advice To A Young Scientist* authored by the Nobel Prize winner Sir Peter Brian Medawar, I collect the answers to the above-listed questions I have received from some of the EATCS Fellows. The real authors of this piece are Mariangiola Dezani-Ciancaglini (University of Turin), Yuri Gurevich (Microsoft Research), David Harel (Weizmann Institute of Science), Monika Henzinger (University of Vienna), Giuseppe F. Italiano (University of Rome Tor Vergata), Scott Smolka (Stony Brook University), Paul G. Spirakis (University of Liverpool, University of Patras and Computer Technology Institute & Press "Diophantus", Patras) and Wolfgang Thomas (RWTH Aachen University), whom I thank for their willingness to share their experience and wisdom with all the members of the TCS community. In an accompanying essay, which follows this one in this issue of the Bulletin, you will find the piece I received from Michael Fellows (University of Bergen).

The EATCS Fellows are model citizens of the TCS community, have varied work experiences and backgrounds, and span a wide spectrum of research areas. One can learn much about our field of science and about academic life in general by reading their thoughts. In order to preserve the spontaneity of their contributions, I have chosen to present them in an essentially unedited form. I hope that the readers of this article will enjoy them as much as I have done.

Mariangiola Dezani-Ciancaglini

The advice I would give to a student interested in TCS is: Your studies will be satisfactory only if understanding for you is fun, not a duty.

To a young researcher in TCS I would say, "Do not be afraid if you do not see applications of the theory you are investigating: the history of computer science shows that elegant theories developed with passion will have eventually long-lasting success."

A research topic that currently excites me is the study of behavioural types. These types allow for fine-grained analysis of communication-centred computations. The new generation of behavioural types should allow programmers to write the certified, self-adapting and autonomic code that the market is requiring.

Yuri Gurevich

Advice I would give to a student interested in TCS Attending math seminars (mostly in my past), I noticed a discord. Experts in areas like complex analysis or PDEs (partial differential equations) typically presume that everybody knows Fourier transforms, differential forms, etc., while logicians tend to remind the audience of basic definitions (like what's first-order logic) and theorems (e.g. the compactness theorem). Many talented mathematicians didn't take logic in their college years, and they need those reminders. How come? Why don't they effortlessly internalize those definitions and theorems once and for all? This is not because those definitions and theorems are particularly hard (they are not) but because they are radically different from what they know. It is easier to learn radically different things — whether it is logic or PDEs or AI — in your student years. Open your mind and use this opportunity!

Advice I would give a young researcher in TCS As the development of physics caused a parallel development of physics-applied mathematics, so the development of computer science and engineering causes a parallel development of theoretical computer science. TCS is an applied science. Applications justify it and give it value. I would counsel to take applications seriously and honestly. Not only immediate applications, but also applications down the line. Of course, like in mathematics, there are TCS issues of intrinsic value. And there were cases when the purest mathematics eventually was proven valuable and applied. But in most cases, potential applications not only justify research but also provide guidance of sorts. Almost any subject can be developed in innumerable ways. But which of those ways are valuable? The application guidance is indispensable.

I mentioned computer engineering above for a reason. Computer science is different from natural science like physics, chemistry, biology. Computers are artifacts, not "naturefacts." Hence the importance of computer science and engineering as a natural area whose integral part is computer science.

A short description of a research topic that excites me at this moment in time (and possibly why) Right now, the topics that excite me most are quantum mechanics and quantum computing. I wish I could say that this is the result of a natural development of my research. But this isn't so. During my long career, I moved several times from one area to another. Typically it was natural; e.g. the theory of abstract state machines developed in academia brought me to industry. But the move to quanta was spontaneous. There was an opportunity (they started a new quantum group at the Microsoft Redmond campus a couple of years ago), and I jumped upon it. I always wanted to understand quantum theory but occasional reading would not help as my physics had been poor to none and I haven't been exposed much to the mathematics of quantum theory. In a sense I am back to being a student and discovering a new world of immense beauty and mystery, except that I do not have the luxury of having time to study things systematically. But that is fine. Life is full of challenges. That makes it interesting.

David Harel

Advice I would give to a student interested in TCS If you are already enrolled in a computer science program, then unless you feel you are of absolutely stellar theoretical quality and the real world and its problems do not attract you at all, I'd recommend that you spend at least 2/3 of your course efforts on a variety of topics related to TCS but not "theory for the sake of theory". Take lots of courses on languages, verification AI, databases, systems, hardware, etc. But clearly don't shy away from pure mathematics. Being well-versed in a variety of topics in mathematics can only do you good in the future. If you are still able to choose a study program, go for a combination: TCS combined with software and systems engineering, for example, or bioinformatics/systems biology. I feel that computer science (not just programming, but the deep underlying ideas of CS and systems) will play a role in the science of the 21st century (which will be the century of the life sciences) similar to that played by mathematics in the science of the 20th century (which was the century of the physical sciences).

Advice I would give a young researcher in TCS Much of the above is relevant to young researchers too. Here I would add the following two things. First, if you are doing pure theory, then spend at least $1/3$ of your time on problems that are simpler than the real hard one you are trying to solve. You might indeed succeed in settling the P=NP? problem, or the question of whether PTIME on general finite structures is r.e., but you might not. Nevertheless, in the latter case you'll at least have all kinds of excellent, if less spectacular, results under your belt. Second, if you are doing research that is expected to be of some practical value, go talk to the actual people "out there": engineers, programmers, system designers, etc. Consult for them, or just sit with them and see their problems first-hand. There is nothing better for good theoretical or conceptual research that may have practical value than dirtying your hands in the trenches.

A short description of a research topic that excites me at this moment in time (and possibly why) I haven't done any pure TCS for 25 years, although in work my group and I do on languages and software engineering there is quite a bit of theory too, as is the case in our work on biological modeling. However, for many years, I've had a small but nagging itch for trying to make progress on the problem of artificial olfaction — the ability to record and remotely produce faithful renditions of arbitrary odors. This is still a far-from-solved issue, and is the holy grail of the world of olfaction. Addressing it involves chemistry, biology, psychophysics, engineering, mathematics and algorithmics (and is a great topic for young TCS researchers!). More recently, I've been thinking about the question of how to test the validity of a candidate olfactory reproduction system, so that we have an agreed-upon criterion of success for when such systems are developed. It is a kind of common-sense question, but one that appears to be very interesting, and not unlike Turing's 1950 quest for testing AI, even though such systems were nowhere in sight at the time. In the present case, trying to compare testing artificial olfaction to testing the viability of sight and sound reproduction will not work, for many reasons. After struggling with this for quite a while, I now have a proposal for such a test, which is under review.

Monika Henzinger

- Students interested in TCS should really like their classes in TCS and be good at mathematics.
- I advice young researchers in TCS to try to work on important problems that have a relationship to real life.
- Currently I am interested in understanding the exact complexity of different combinatorial problems in P (upper and lower bounds).

Giuseppe F. Italiano

The advice I would give to a student interested in TCS There's a great quote by Thomas Huxley: "Try to learn something about everything and everything about something." When working through your PhD, you might end up focusing on a narrow topic so that you will fully understand it. That's really great! But one of the wonderful things about Theoretical Computer Science is that you will still have the opportunity to learn the big picture!

The advice I would give a young researcher in TCS Keep working on the problems you love, but don't be afraid to learn things outside of your own area. One good way to learn things outside your area is to attend talks (and even conferences) outside your research interests. You should always do that!

A short description of a research topic that excites me at this moment in time (and possibly why) I am really excited by recent results on conditional lower bounds, sparkled by the work of Virginia Vassilevska Williams et al. It is fascinating to see how a computational complexity conjecture such as SETH (Strong Exponential Time Hypothesis) had such an impact on the hardness results for many well-known basic problems.

Scott Smolka

Advice I would give to a student interested in TCS Not surprising, it all starts with the basics: automata theory, formal languages, algorithms, complexity theory, programming languages and semantics.

Advice I would give a young researcher in TCS Go to conferences and establish connections with more established TCS researchers. Seek to work with them and see if you can arrange visits at their home institutions for a few months.

A short description of a research topic that excites me at this moment in time (and possibly why) Bird flocking and V-formation are topics I find very exciting. Previous approaches to this problem focused on models of dynamic behavior based on simple rules such as: Separation (avoid crowding neighbors), Alignment (steer towards average heading of neighbors), and Cohesion (steer towards average position of neighbors). My collaborators and I are instead treating this as a problem of Optimal Control, where the fitness function takes into account Velocity Matching (alignment), Upwash Benefit (birds in a flock moving into the upwash region of the bird(s) in front of them), and Clear View (birds in the flock having unobstructed views). What's interesting about this problem is that it is inherently distributed in nature (a bird can only communicate with its nearest neighbors), and one can argue that our approach more closely mimics the neurological process birds use to achieve these formations.

Paul G Spirakis

My advice to a student interested in TCS Please be sure that you really like Theory! The competition is high, you must love mathematics, and the money prospects are usually not great. The best years of life are the student years. Theory requires dedication. Are you ready for this?

Given the above, try to select a good advisor (with whom you can interact well and frequently). The problem you choose to work on should psyche you and your advisor!

It is good to obtain a spherical and broad knowledge of the various Theory subdomains. Surprisingly, one subfield affects another in unexpected ways.

Finally, study and work hard and be up to date with respect to results and techniques!

My advice to a young researcher interested in TCS Almost all research problems have some difficulty. But not all of them are equally important! So, please select your problems to solve carefully! Ask yourself and others: why is this a nice problem? Why is it interesting and to which community? Be strategic!

Also, a problem is good if it is manageable in a finite period of time. This means that if you try to solve something open for many years, be sure that you will need great ideas, and maybe lots of time! However, be ambitious! Maybe you will get the big solution! The issue of ambition versus reasonable progress is something that you must discuss with yourself!

It is always advisable to have at least two problems to work on, at any time. When you get tired from the main front, you turn on your attention to the other problem.

Try to interact and to announce results frequently, if possible in the best forums. Be visible! It is important that other good people know about you. “Speak out to survive!”

Study hard and read the relevant literature in depth. Try to deeply understand techniques and solution concepts and methods. Every paper you read may lead to a result of yours if you study it deeply and question every line carefully! Find quiet times to study hard. Control your time!

A field that excites me: the discrete dynamics of probabilistic (finite) population protocols Population Protocols are a recent model of computation that captures the way in which complex behavior of systems can emerge from the underlying local interactions of agents. Agents are usually anonymous and the local interaction rules are scalable (independent of the size, n, of the population). Such protocols can model the antagonism between members of several “species” and relate to evolutionary games.

In the recent past I was involved in joint research studying the discrete dynamics of cases of such protocols for finite populations. Such dynamics are, usually, probabilistic in nature, either due to the protocol itself or due to the stochastic nature of scheduling local interactions. Examples are (a) the generalized Moran process (where the protocol is evolutionary because a fitness parameter is crucially involved) (b) the Discrete Lotka-Volterra Population Protocols (and associated Cyclic Games) and (c) the Majority protocols for random interactions.

Such protocols are usually discrete time transient Markov Chains. However the detailed states description of such chains is exponential in size and the state equations do not facilitate a rigorous approach. Instead, ideas related to filtering, stochastic domination and Potentials (leading to Martingales) may help in understanding the dynamics of the protocols.

Some such dynamics can describe strategic situations (games): Examples include Best-Response Dynamics, Peer-to-Peer Market dynamics, fictitious play etc.

Such dynamics need rigorous approaches and new concepts and techniques. The ‘traditional’ approach with differential equations (found in e.g. evolutionary game theory books) is not enough to explain what happens when such dynamics take place (for example) in finite graphs with the players in the nodes and with interactions among neighbours. Some main questions are: How fast do such dynamics converge? What is a 'most probable' eventual state of the protocols (and the computation of the probability of such states). In case of game dynamics, what is the kind of 'equilibria' to which they converge? Can we design 'good' discrete dynamics (that converge fast and go to desirable stable states ?). What is the complexity of predicting most probable or eventual behaviour in such dynamics?

Several aspects of such discrete dynamics are wide open and it seems that the algorithmic thought can contribute to the understanding of this emerging subfield of science.

Wolfgang Thomas, "Views on work in TCS"

As one of the EATCS fellows I have been asked to contribute some personal words of advice for younger people and on my research interests. Well, I try my best.

Regarding advice to a student and young researcher interested in TCS, I start with two short sentences:

- Read the great masters (even when their h-index is low).
- Don't try to write ten times as many papers as a great master did.

And then I add some words on what influenced me when I started research — you may judge whether my own experiences that go back to "historical" times would still help you.

By the way, advice from historical times, where blackboards and no projectors were used, posed in an entertaining but clearly wise way, is Gian-Carlo Rota's paper "Ten Lessons I Wish I Had Been Taught" (`http://www.ams.org/notices/199701/comm-rota.pdf`). This is a view of a mathematician but still worth reading and delightful for EATCS members. People like me (68 years old) are also addressed — in the last lesson "Be Prepared for Old Age"...

Back in the 1970's when I started I wanted to do something relevant. For me this meant that there should be some deeper problems involved, and that the subject of study is of long-term interest. I was attracted by the works of Büchi and Rabin just because of this: That was demanding, and it treated structures that will be important also in hundred years: the natural numbers with successor, and the tree of all words (over some alphabet) with successor functions that represent the attachment of letters.

The next point is a variation of this. It is a motto I learnt from Büchi, and it is a warning not to join too small communities where the members just cite each other. In 1977, when he had seen my dissertation work, Büchi encouraged me to continue but also said: Beware of becoming member of an MAS, and he explained that this means "mutual admiration society". I think that his advice was good.

I am also asked to say something about principles for the postdoctoral phase. It takes determination and devotion to enter it. I can say just two things, from my own experience as a young person and from later times. First, as it happens with many postdocs, in my case it was unclear up to the very last moment whether I would get a permanent position. In the end I was lucky. But it was a strain. I already prepared for a gymnasium teacher's career. And when on a scientific party I spoke to Saharon Shelah (one of the giants of model theory) about my worries, he said "well, there is competition". How true. So here I just say: Don't give away your hopes — and good luck. The other point is an observation from my time as a faculty member, and it means that good luck may be actively supported. When a position is open the people in the respective department do not just want a brilliant researcher and teacher but also a colleague. So it is an important advantage when one can prove that one has more than just one field where one can actively participate, that one can enter new topics (which anyway is necessary in a job which lasts for decades), and that one can cooperate (beyond an MAS). So for the postdoc phase this means to look for a balance between work on your own and work together with others, and if possible in different teams of cooperation.

Finally, a comment on a research topic that excites me at this moment. I find it interesting to extend more chapters of finite automata theory to the infinite. This has been done intensively in two ways already — we know automata with infinite "state space" (e.g., pushdown automata where "states" are combined from control states and stack contents), and we know automata over infinite words (infinite sequences of symbols from a finite alphabet). Presently I am interested in words (or trees or other objects) where the alphabet is infinite, for example where a letter is a natural number, and in general where the alphabet is given by an infinite model-theoretic structure. Infinite words over the alphabet $\mathbb{N}$ are well known in mathematics since hundred years (they are called points of the Baire space there). In computer science, one is interested in algorithmic results which have not been the focus in classical set theory and mathematics, so much is to be done here.

Are you interested in theoretical computer science? (How not???) I have some advice for you

Michael Fellows
Department of Informatics, University of Bergen
michael.fellows@uib.no

Long ago, in the time of MEGA-Math (sponsored by the Los Alamos National Laboratories) Nancy Casey and I were doing some of those activities that are now better known as *Computer Science Unplugged!* activities (both easily googled) that are designed to present mathematical ideas that are foundational to computer science in concrete ways accessible to school-age children. We were visiting the first-grade classroom of legendary teacher Prudy Heimsch in Moscow, Idaho. Wondering how these concrete activities basically about theoretical computer science fit with her classroom objectives, we asked her, "Do you have any particular objectives in mind for your six-year-olds?"

Prudy answered without hesitation, *Yes I do! I have three goals for them:*

I want them to learn to communicate.

I want them to learn to think about their own thinking.

I want them to be able to formulate a project of their own devising, and carry it through to completion.

These core objectives will serve until graduate school and beyond! This being the case, here is my advice for students.

The call for these contributions of advice from the EATCS Fellows, asks for the discussion to be organized as advice for four different levels of education: School level: meaning primarily High School, University level, PhD students, Young researchers beyond the PhD. But what about the kids? The burning heart of curiousity is 6-7-8-9-10 year-olds! Attend to the fire.

Primary School. We were in love with Einstein when we were ten years old. We wanted to be theoretical physicists when we grow up. There was no chance for Relativity to be in the school curriculum. But there was a chance, in those olden days, to get a bit of what they used to call "enrichment". Someone would come to the school and tell us exciting stuff that was not in any lesson plan.

If you are a little kid, with any inkling of what theoretical computer science is about, or you just want to know, my advice to you is to demand **enrichment** to theoretical computer science, so that you can be exposed to exciting contemporary science, heart of modern civilization.

That's the advice if you are in the lower grades: **demand some exposure to theoretical computer science**. If your parents work in IT: **demand** that they visit your classroom, or that they arrange for one of their friends to come to your classroom.

Take charge of educating yourself. There is plenty of stuff on *Computer Science Unplugged!* at `http://csunplugged.org/`.

Much of my advice for High School and beyond is aimed at you as well. Explore questions. Fill notebooks with drawings, ideas, speculations, guesses, investigations. Look for the big picture, and then try to see details and how details fit together to make the big picture.

High School. Explosion of interests and engagements! Bring it on! Later you may appreciate all the ironies involved. Cultivate a passionate interest in literature. You cannot be a great scientist without it–to a first approximation. There is some weird correlation between physical and intellectual courage. Go surfing! Who is to advise people at this flowering age beyond th emselves? OK, pick an intellectually ambitious book about TCS, stick it in your backpack and GO. Mine was Goldblatt's book on Topoi, to the High Sierras of California. Just go, go, go! Develop mentors! Famous scientists (and younger ones) are easier to approach than you might imagine — just email, or write a letter on paper. There are tons of open problems — just join in. Do not waste too much time with computers. If you are interested in theoretical computer science, you are going to need a lot of mathematics, the ultimate magical mind game, with power over most everything going on, except for love.

University Student. There is almost no chance for a serious research career in theoretical computer science without solid foundations in Mathematics.

Learn math! If you are a college student, a double major in both Mathematics and Computer Science is a good option. If you must choose, choose Math. Mathematics is somehow ever-metacognitive, across all fields.

Join in! If you are an undergraduate student studying Mathematics with an interest in Computer Science, then you should join both the *American Mathematical Society* and the *Association for Computing Machinery*. I did that when I went to college after the Vietnam War, and although I didn't have much money at the time, it was the best investment I ever made. Publications will arrive to your mailbox describing exciting contemporary developments in these fields. Read about them, even if at the beginning, you don't really understand much of what is going on. Fields Medal winner Michael Freedman, who was a mentor to me, once told me about how he got into Mathematics as an undergraduate by browsing research journals and becoming obsessed with trying to figure out what they were on about.

Do that!

Educate yourself! Do not expect the university or the high school or the elementary school to educate you. At best, realistically, they won't get too much in your way. The school curriculum is inherently way too slow to keep up. You must depend on forming and making support groups of friends, finding mentors (easier than you might think), and taking things into your own hands.

Do research! It is never too early to take up a research project appropriate to your level. Mentors will happily help you. Exploring questions that nobody knows the answer to is *really fun*. Maybe surfing a really big wave is competitive in the short term, but doing science is fun and gratifying. If you manage to publish research as a high school student, which is a perfectly reasonable objective, this will open doors when you apply to the universities. If you manage to publish research as a university student, this will open doors when you apply to graduate school. Working on a real project is the best way to learn.

PhD Student in Computer Science. During a wonderful visit with the research group of Prof. Jianxin Wang at Central South University, Changsha, China, a student engaged me over dinner asking for advice and this is the result of our conversation. There are two different advices. One is for science and one for career. Often, they overlap.

SCIENCE

• Do not trust your advisor. Watch out for "mini–Me". Be prepared to sniff it out and bolt. Your interests do not align. Your advisor is (especially if young) almost always desperately interested in advancing their own career, and you should study the blues. There are exceptions, but they tend to be rare. Your true mission is to pioneer new and outlandish angles, ideas and kinds of questions, and be prepared to defend your own new directions of research.

• Put research first. Learn by doing. Try things out. Dream big.

• Work on more than one problem at a time, or in more than one research direction. Have several functional advisors.

• Don't be shy. Write to other scientists and to the authors of papers you have read.

• Choose your problems. Some problems have important applications, while others have the potential to build theory. Some people are natively problem solvers with sharp tools and others are theory builders with a big picture view, although probably all are a bit of both.

• Visit. This is part of, "Don't be shy". Visit other research teams. Learn who else is working in your areas. Get to know the other students and leaders.

• Pick good partners. Good research partners inspire each other to keep going past the 'finish line' and get the job completely done and on time.

• Make friends in other fields. Each field has its own vocabulary and solving techniques.

CAREER

There are two targets that you may be working towards: science and career. Both are important and there is no shame in favouring one over the other at various points in time—or in point of your interest or career (there may be ebb and flow).

• Enjoy writing for grants. It may seem strange to suggest enjoyment of grantwriting, but it is an opportunity to hone your writing skills, explain what you care about to people who may not be in the same area, (often they are not), to clarify in your own mind your plans and aspirations, rethink and reconnect with your collaborators. Once you have written a solid grant proposal, you may want to give away the ideas to others who are seeking funding from other sources. Good ideas deserve to be funded.

• Serve the reader. We have all encountered speakers and writers with an agenda of making themselves look intelligent and knowledgeable by using words and phrases that obfuscate the issues. Some writers give multiple citations to obscure references. Our job is to help the reader in every way possible to understand the sometimes arcane material we are offering.

• Story is central. Story is a bigger force than science. Everybody lives by stories. They are a primal force. In mathematics, we add formalism. We have equations that lead to solutions but story has its own logic. Find the story in what you are telling and presenting. This will help the listener meet you more than half-way.

• Generously credit everyone else's work. Fiercely defend your own IP.

Postdoctoral Researcher and Young Professor. The advice above for PhD students in Computer Science is appropriate for you, as well.

• When writing papers, generously credit previous and related work. Do not make the mistake of citing only FOCS, STOC and SODA papers. Your proper goal is to *serve the reader* not *impress the reader*.

• Now is your time to liberate your curiosity! Hedge your bets and work on more than one project. Have several functional advisors. Communicate with and visit other teams in other universities.

• Do stuff with young people. – Refreshes your morale (and amazes your colleagues and is very easy to do.)

– Good for your kids.

– Good for everybody, really.

– Good for science, government and industry because the effort to clearly communicate your ideas brings out new basic questions. – This is NOT about university recruitment!

• Think about moving (geographically)! Don't be overly concerned about moving to a "small" place. When I was a young professor, I joined a small, fairly remote university for family reasons. I was the only research professor, and this meant that I got ALL the research budget, which was considerable since the university got almost all the state's research budget. You may become a "big fish in a small pond" and be able to influence important research directions in your area.

Horizons.

The original call for these contributions of advice from the EATCS Fellows, asked for: *A short description of a currently favourite research topic.*

Many parameterised problems are FPT and the toolkit for approving such results is quite varied. Yet in some sense, because a parameterised problem is FPT if and only if there is a P-time kernelization algorithm, one can say that proving the best possible P-time kernelization bound is the canonical issue. Many FPT kernelization results can be proved by neatly structured and algorithmically well-behaved argumentation. This programmatic point of view is advocated in the paper *FPT is P- time Extremal Structure Theory: the Case of* MAX LEAF which gives a good example. I am interested in axiomatizing and exploring the limits and meta-theorems about *groovy FPT*: kernelization results where everything in sight is polynomial time or has a polynomial time interpretation. From such well-structured results, one can canonically derive P-time approximation algorithms, and canonically associated inductive gradients of interest to local search algorithms and greedy heuristics, thus neatly connecting several different basic algorihmic issues. There are also concrete examples showing that groovy FPT is a proper subset of FPT, so that this is a really exciting and fundamental direction in FPT research.

Contributions by EATCS Award Recipients

Interview with Stephen Brookes and Peter W. O'Hearn Recipients of the 2016 Gödel Prize

Luca Aceto
ICE-TCS, School of Computer Science,
Reykjavik University
`luca@ru.is`

Stephen Brookes (Carnegie Mellon University, USA) and Peter W. O'Hearn (Facebook and University College London, UK) are the recipients of the 2016 Gödel Prize for their invention of Concurrent Separation Logic, as described in the following two papers:

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

Quoting from the citation for the prize:

> "Concurrent Separation Logic (CSL) is a revolutionary advance over previous proof systems for verifying properties of systems software, which commonly involve both pointer manipulation and shared-memory concurrency. For the last thirty years experts have regarded pointer manipulation as an unsolved challenge for program verification and shared-memory concurrency as an even greater challenge. Now, thanks to CSL, both of these problems have been elegantly and efficiently solved; and they have the same solution."

As highlighted in the citation by the 2016 Gödel Prize committee, the impact of CSL has been extraordinary, both from a theoretical and practical viewpoint. CSL has led to a wealth of follow-up research over the last decade and "its simplicity and structure also facilitates automation. As a result numerous tools and techniques in the research community are based on it and it is attracting attention in companies such as Microsoft, Facebook and Amazon."

In order to celebrate the award of the Gödel prize to this path-breaking work and to allow the research community in theoretical computer science at large to appreciate the origin of the ideas that led to their invention of Concurrent Separation Logic, I interviewed Stephen Brookes (abbreviated to SB in what follows) and Peter W. O'Hearn (referred to as PO in the text below) via email. I hope that the readers of the Bulletin of the EATCS will enjoy reading the text of the interview as much as I did.

1 The interview

LA: You are receiving the Gödel Prize 2016 for your invention of Concurrent Separation Logic (CSL), which, quoting from the prize citation, is "a revolutionary advance over previous proof systems for verifying properties of systems software, which commonly involve both pointer manipulation and shared-memory concurrency." Could you briefly describe the history of the ideas that led to the invention of CSL, what were the main inspirations and motivations for its invention and how CSL advanced the state of the art?

PO: After John Reynolds and I and others had done the initial work on separation logic for sequential programs it made sense to consider concurrency, just based on the idea of using the logic to keep track of the separation of resources used by different processes or threads. In the summer of 2001 I devised initial proof rules to do just that, by adapting an approach of Hoare to reasoning about concurrency from Hoare logic to separation logic.

This first step seemed straightforward enough, but then it hit me that we could use the logic to track dynamically changing partitions rather than the static partitioning in Hoare's approach. I used a little program, the pointer-transferring buffer, to explore this idea, and I made a program proof in which the fact that something was allocated seemed to move from one process to another... during the proof. The pointer itself was copied, but the fact that it was allocated (and hence the right to dereference it) was given up by the sending process. I began talking about "knowledge" or "ownership" transferring from one process to another. The striking thing was that there was no explicit concept of ownership in the logic or proof rules, but that this transfer seemed to be encoded in the way that the proofs of the processes worked. I circulated an unpublished note on proving the pointer-transferring buffer in August of 2001, and then a longer note in January 2002; these documents got a lot of attention from people working on separation logic.

It quickly became clear that quite a few concurrent programs would have much simpler proofs than before. Modular proofs were provided of semaphore programs, of a toy memory manager, and programs with interacting resources. I got up a huge head of steam because it seemed as if the logic could explain the way that synchronisation had been used in the fundamental works on concurrent programming by Dijkstra, Hoare and Brinch Hansen. For example, in the paper that essentially founded concurrent programming, Dijkstra in 1965 (*Co-operating Sequential Processes*, still the most important paper in concurrency) had explained that the point of synchronisation was to enable programmers to avoid minute considerations of timing, to simplify reasoning. Brinch Hansen had hammered into me the importance of speed independence and resource separation for simplifying thinking about concurrent processes when I was his colleague at Syracuse in the 1990s, and what he said to me seemed to be mirrored in the proofs in this early concurrent separation logic. And although I had never written a paper on concurrency, I was opinionated: I thought that work in the theory of concurrency was often missing the mark because while it could describe semaphores and other synchronisation primitives, it did not explain their significance because it did not connect back to simplifying reasoning; which was their whole point. Hoare had made important steps in various points in his work, but it seemed as if we could go much further armed with the separating conjunction.

So, years of thinking about these issues seemed poised to come together at once in this concurrent separation logic. But, for all my opinionated excitement, I ran into a blocker of a problem: I wasn't able to prove soundness of my proof rules. And it was the very feature which gave rise to the unexpected power, the ownership or knowledge transfer, that made soundness non-obvious. I worked hard on this problem for several months in the second half of 2001 and early in 2002, and got nowhere. Reynolds, Yang, and Calcagno, all semantics experts, also looked at the problem. Finally, I admitted to myself that it was technically beyond my expertise in concurrency theory and that I needed help. Luckily, I knew where to turn. Steve Brookes and I had never worked together before, but knew one another from way back. He was the external examiner of my 1991 PhD thesis, I was well aware of his fundamental work with Hoare and Roscoe on the foundations of CSP, and he had recently produced striking results on full abstraction for shared-memory parallelism, what I though of as the most impressive theoretical results in semantics of concurrency at the time. What is more, Steve had built up a powerful repertoire of techniques for proving properties of concurrent programming languages. So, sometime in 2002, I picked up the phone and gave him a call: "Steve, I have a problem!".

SB: As Peter said, he called me out of the blue. I knew Peter well, having served as the external examiner on his PhD thesis, and I had kept in contact with him after he took up his first academic positions at Syracuse and at Queen Mary (University of London). We were in fairly regular email contact, but he didn't normally phone me from England; I knew this must be important. I sat in my office at Carnegie

Mellon University in Pittsburgh, intrigued and fascinated to hear his ideas and excited by the challenge he was offering me. I think he also spoke by phone at a different time to John Reynolds, whose office was right next to mine. We all agreed that the best plan was for Peter to come to Pittsburgh and give a talk, after which we would do some brainstorming. That was how it started, from my viewpoint. Peter came to CMU in March of 2002 and gave a talk on his new logic. Peter was proposing a subtle and clever combination of key ideas from separation logic and a much earlier Owicki-Gries logic for shared-memory concurrency, itself based on ideas appearing in a classic paper of Tony Hoare (*Towards a theory of parallel programming*). Superficially the new logic was both very simple and also very perplexing. The Hoare-style logic has a simple rule for parallel composition that combines pre- and post-conditions using conventional conjunction, and a rule for conditional critical regions (essentially, binary semaphores) that makes use of "resource invariants". The inference rules ensure that a provable program obeys what has come to be known as a "rely-guarantee" discipline: each process assumes that whenever it acquires a semaphore the relevant invariant holds, and guarantees that the invariant holds again when releasing the semaphore. However, the earlier logic is not sound for programs using pointers, because of the possibility of aliasing. On the other hand separation logic was originally formulated for reasoning about sequential programs using pointers, but lacked rules for shared-memory concurrency and (until then) it was not clear how to generalize. Peter introduced a radically simple and elegant way to combine the two: an overly simplistic summary is that he allows separation logic formulas as pre- and post-conditions (and resource invariants) and strategically replaces certain occurrences of conjunction in the Hoare-Owicki-Gries rules with the separating conjunction operator from separation logic. Of course this naive characterization glosses over some profound issues: in Peter's approach the invariants and pre- and post-conditions are not really talking about the "global" state, but serve to specify the "footprint" of a program component, just the piece of state on which that component acts. I think this was the first time I saw use of terms such as "footprint" and "ownership transfer", notions which now seem very intuitive but as yet had no formal semantic counterpart. (Until this point, I think it is fair to say, semantic models for concurrent languages had typically dealt entirely with global state, and it was conventional wisdom that it was already hard enough to find a decent semantics for simple shared-memory programs, let alone try to incorporate mutable state, heap cells, allocation and deallocation.) My challenge was to develop a semantic model robust and flexible enough to handle the combination of concurrency and pointers, in which such notions as ownership transfer could be properly formalized: to build a foundation on which to establish soundness of Peter's proposal. Furthermore, prior semantic models for concurrent languages had pretty much ignored race conditions, typically by assuming that assignments were executed atomically so that races never happen. While that worked well for "simple" shared-memory it was clearly an insufficiently sophisticated way to cope with mutable state and the more localized view of state that is so fundamental in Peter's approach.

During this visit we basically barricaded ourselves in a room with Reynolds and Calcagno, dropping all other distractions and tossing ideas around in an attempt to lay out a groundplan, exploring the apparent benefits of the new logic while probing for limitations and possibly even counterexamples. John's wife Mary recalls very intense discussions at the Reynolds's household, walls covered with sticky paper, only breaking for meals. Peter enunciated what became known as the Separation Principle: "at all stages, the global heap is partitioned into disjoint parts...", another way to articulate the idea of "ownership transfer". Again it is easy to express these notions informally, but it turned out to be surprisingly tricky to encapsulate them semantically, and Peter was right to be cautious.

I remember marveling at the ease with which Peter was able to articulate, with simple-looking little programs like a one-place buffer, the kinds of problem that arise when concurrent threads manipulate the heap. And the logic seemed elegantly suited for reasoning about the correctness of such programs, with the decided advantage that provable programs are guaranteed to be free of data races. (This was also true of the earlier Hoare-Owicki-Gries logic, but only in the much more limited setting of "simple" shared-memory without pointers.)

The use of separation in the logic neatly embodies a kind of disciplined use of resources in programs. Yet it was by no means clear how to formalize these ideas, and Peter was quite forthright about his unwillingness to publicize the ideas until it had been demonstrated that it all made sense. He had circulated an "unpublished manuscript" with the title *Notes on separation logic for shared-variable concurrency*, dated January 2002. Meanwhile I was excited to have a challenging problem to work on, and intense interactions continued between Peter, John and me as our understanding evolved.

I had plenty of experience in developing denotational semantic models for (simple) shared-memory programs and also for communication-based languages such as CSP. Just like the earliest denotational accounts of concurrency (dating back to David Park in the late 70's) the most widespread and generally most adaptable approach was to use traces (or sequences of actions) of some kind; I had introduced "transition traces", built from steps that represent (finite sequences of) atomic operations by a process, with gaps allowing for state changes made by the "environment". I think that transition trace semantics is what Peter was referring to, when he mentioned full abstraction, but I should emphasize that this model was tailored specifically to simple shared-memory and I could not see an obvious way to adapt it to incorporate pointers. I did try! My more recent focus had been on "action traces", in which steps represent atomic actions but the details concerning state (when an action is enabled, and what state change it causes) are kept apart: a process denotes a set of action traces, and one can then plug in a model of state, give an interpretation of actions as state transformers, and be able to reason rigorously about program execution. It seemed to me that action traces offered the best basis for expansion to incorporate pointers: by augmenting the "alphabet" of actions to include heap look-up, update, allocation and deallocation. It should also be quite natural to treat semaphore operations (for acquiring and releasing) as actions. It took until some time in 2003 for me to iron out the technical details and be able to explain the key definitions and results clearly enough to be ready for publication.

In retrospect I regard this period of a couple of years as probably the most exciting and stimulating sustained research in which I have been involved. I am immensely grateful to have had the opportunity to work with Peter on this project. And throughout all of this I was strongly influenced by John Reynolds, whose good taste, deep originality of thought and sheer intellectual inquisitiveness served to keep me focussed. Echoing Peter's reluctance to publish without being sure, I always said to myself that I wouldn't be sure until I could convince John (and Peter!). John and I had lunch together almost every day, and I camped out in his office whenever I had an idea that needed a sounding board. He prompted me to always seek clearer explanations, isolate the key concepts and make the right definitions, and strive for generality. I remember in particular that at one point, after several weeks trying to figure out how to extend action traces, I told John that I might be able to give a semantics in which (only) provable programs would be definable. My naive idea was to modify the way that parallel composition gets modelled to keep track of the preservation of resource invariants explicitly and treat any violation as a "disaster". I recall John upbraiding me gently about this plan; in his view, one should develop a semantics that is "agnostic" about any intended logic, and instead built to express computational properties of programs in general terms. He did agree that we needed a semantics that reported the potential for race conditions, an idea that is echoed in his own work. Having found such a semantics, it ought then to be possible to show using the semantics that programs provable in the logic are indeed race-free. In essence, that's what happened. But it didn't happen overnight, and the path from start to end involved a few detours and dead-ends before finding a robust solution.

There was a crucial role in this played by the concept of "precise" assertion. I will hand back to Peter to make a few remarks on that.

PO: Yes, a memorable moment came when John poked his head into my office one day: "want some bad news?". He showed me an example where the proof rule for critical regions together with the usual Hoare proof rule for using conjunction in the postcondition lead to an inconsistency. This problem had nothing to do with concurrency per se, but rather was about the potential indeterminacy of the "angel" affecting ownership transfer. (We had been attempting to get a handle on ownership transfer by couching it in adversarial terms, involving an "angel" making program choices and a "demon" trying to invalidate invariants or induce race conditions.) The problem also arose in a sequential setting, in proof rules for modules that I was working on with John and Yang (which we eventually published in *Separation and Information Hiding*, in POPL'04). In any case, I quickly proposed a concept of "precise" assertion, one that unambiguously picks out an area of storage, as a way to get around the problems cause by the indeterminacy of the angel, and this concept is used in the resource invariants in concurrent separation logic.

This indicates what a subtle problem we were up against. Some experienced semantics researchers had been working on soundness of CSL and of information hiding and there lurked a problem that none of us had spotted. We were lucky that John Reynolds was not only keeping us honest by tracking our progress, but thinking deeply about these problems himself. Concurrent separation logic owes a lot to John's brilliance.

That being said, this problem with the ownership angel was not the biggest hurdle Steve had to get over. Setting up the concurrency semantics and identifying the right properties to prove to get inductions to go through was just hard. Once he had done it, others were able to generalize and sometimes (arguably) simplify his proof method; e.g., in work of Calcagno, Yang, Gotsman, Parkinson, Vafeiadis, Nanevsky, Sergey and others, and some of it drops the precision requirement (but also the conjunction rule). But, from my perspective, Steve solving this problem for the first time was difficult, and important. I like to say: he saved my logic!

LA: I noticed that you each published short versions of your papers in the same conference, CONCUR'04, and then again the long versions appeared in the same journal issue in TCS'07. How did that come about?

SB: My email to Peter and John, announcing that I had found the "right" semantics and that the concurrency rules can be shown to be sound when resource invariants are chosen to be precise, dates from early June 2003. (I also showed that the rules remain sound when invariants are chosen to be "supported", so that in any state with a sub-heap satisfying an invariant there is a unique minimal sub-heap with that property.) This was obviously very exciting and I headed over to England to spend some time in London with Peter. Peter organized informal discussion meetings (called "yak sessions" to distinguish from full-blown seminars) and Peter and I both gave presentations there on CSL. Philippa Gardner witnessed these presentations, and she was so enthusiastic that she decided (as organizer for the CONCUR 2004 conference in the following year) to invite us to give back-to-back hour-long tutorials at that meeting. That's how the original short versions of our papers ended up at the same conference. We felt honoured to be asked, effectively, to give tutorial invited lectures on as-yet unpublished work! And thanks to Philippa for her part in encouraging this scenario.

We published the full, journal-length versions together in TCS as tributes to John on his 70th birthday. The Festschrift volume appeared as a book, under the TCS imprimatur, edited by Peter along with Olivier Danvy and Phil Wadler, in 2007. There's an amusing story that Peter reminded me of. One day Philippa was walking with me, John and Peter, some time before the CONCUR short versions were finalized. John was berating Peter for what he perceived perhaps as dilatoriness in not submitting long versions to POPL or some other conference instead of preparing the journal-length versions; he couldn't understand why it was taking us so long to get around to it. What he did not realize was that the plans were already afoot for his birthday festschrift, and we had already agreed to publish there. After all, what better way to acknowledge the profound influence John had on the work. Philippa leaned over to Peter and whispered that we couldn't tell him because those (long) papers are for his "bloody festschrift".

LA: In your opinion and experience, how important are modularity and compositionality in proving properties of programs and systems?

SB: I think the benefits of modularity — in particular, allowing more "localized" reasoning by considering each component largely in isolation from the others — have been touted right from the early days. For example Dijkstra was in favor of "loose coupling", and argued that the art of parallel program design was to ensure that processes can be regarded almost as independent, except for the (ideally small number of) places where they synchronize. The claim was, and continues to be, that this would improve our ability to manage the sheer complexity caused by interactions between concurrent threads. For similar reasons, and dating back to Hoare logic (1969!), we seek compositional proof systems for proving program properties. In a compositional proof system one can derive correctness properties of a program by reasoning about program components individually, and the inference rules show how the properties of components determine the behaviour of the whole program. In short, compositional means "syntax-directed", and again a major desire is to exploit syntax-directed analysis to tame the combinatorial explosion. But to design a compositional logic for a specific programming language, for use in establishing a specific class of program behaviour, you need to start with a suitably chosen assertion language — one for which compositional reasoning like this is even possible. This is particularly difficult for concurrent programs, since it has long been known that Hoare-style partial correctness assertions about a multi-threaded program cannot be deduced simply on the basis of partial correctness properties of individual threads. A partial correctness assertion of form $\{p\}\ c\ \{q\}$ says that every terminating execution of c from an initial state satisfying p ends in a state satisfying q. In the early Hoare-style logics for shared-memory programs (Hoare-Owicki-Gries, as mentioned earlier) assertions look just like conventional partial correctness but are interpreted semantically as expressing a much more sophisticated property, allowing for the program to be running in an "environment" of other processes. It has become common to describe this interpretation in terms of "rely/guarantee". I think one of the key ingredients in my semantic model is that it allows formalization of the kind of ownership transfer discipline inherent in Peter's inference rules. Turning this on its head, you could say that the semantic model supports compositional reasoning about programs whose correctness is justified by appeal to Peter's separation principle and the ownership discipline. I'll let Peter step in here too, as I'm sure he feels strongly about compositionality as a virtue. Maybe he can say something about monitors as well, which I know were a strong motivating factor in how he came up with the inference rules.

PO: Generally speaking, when proving a program it is possible in principle to construct a global proof, one that talks about the global state of the system. But global proofs tend to be much harder to maintain when the program is changed. And programs are constantly being changed in the real world: the world won't accept prove-it-and-forget-it proof efforts, verification needs to be active and move with the programmers. This, even more than efficiency considerations in constructing proofs at the outset, is the strongest reason for wanting modularity.

Separation logic has just provided a theory that often matches the intuitive modularity that comes up in data structure designs. The degree of modularity in proofs that have been done has been surprising. For instance, when I was thinking about proof rules for CSL my first idea was to axiomatize monitors (class-like abstractions for concurrency due to Brinch Hansen and Hoare), because I thought that low-level primitives like semaphores were too unstructured and that modular proofs would be impossible. Of course I knew that monitors could simulate semaphores, but I didn't expect to find nice proofs of semaphore programs. It therefore came as a bit of a shock when I was able to provide very local independent reasoning about semaphores in some nontrivial examples.

But, while modularity is important, you should be careful not to try to take it too far. For instance, sometimes multiple resources participate together in the implementation of a data abstraction, like the use of several locks in hand-over-hand locking on linked lists. Vafeiadis and Parkinson have some lovely work on RGSep, a descendant of the original concurrent separation logic, in which they show how you can describe the effects of operations in a very local way, but where the description sometimes involves two locks and a bit of a linked list, rather than only one lock; you would just be causing yourself trouble if you tried to formulate everything in terms of independent reasoning about the individual locks in this case. So I like to think that you should make your specifications and proofs as modular as is natural, but not more so; logic should not block making specifications and proofs modular, but neither should it force you to shoehorn your descriptions into a fixed granularity of modularity.

Some of the work that follows on from mine and Steve's work is very flexible in the degree of abstraction that can be given to the way that state is composed and decomposed. For instance, work of Nanevski, Sergey, Dreyer, Birkedal and others is all based on proof methods which allow the granularity of modularity or separation to be chosen, but specifying a partial commutative monoid of composition (in place of the standard separation logic monoid of heaplets and disjoint union).

LA: What are some of the main developments since your papers appeared.

PO: First let me mention developments in theory. To me, the most surprising has been the demonstration that the most basic principles of concurrent separation logic, particularly independent reasoning about threads using the separating conjunction, cover a much broader range of situations than we ever expected. There have been proofs of fine-grained locking and non-blocking concurrency and cases that involve interference and general graph structures, what might have been though of as bad cases originally for separation logic.

SB: We should also mention generalizations based on permissions (for example, Boyland's account of fractional permissions). In particular this path led to versions of CSL capable of dealing naturally with concurrent reads, and to some very elegant program proofs (due to Peter with Calcagno and Bornat) in which the correctness of the program depends on a permissive form of ownership transfer.

PO: Interestingly, the unexpected power of this is based on what you might call "non-standard models" of separation logic; I mean this by analogy with the usual situation in logic, where a theory (e.g. reals, or integers) has an intended model, but then additional non-standard models of the same axioms. The proof theory can then accomplish unexpected things when applied to the non-standard models. The standard model of separation logic is the original model based on splitting portions of the heap, or heaplets. There are lots of other models stemming from the "resource semantics" of bunched logic invented by David Pym (based on having a partial commutative monoid of possible worlds). The surprise is that some of these nonstandard models involve composing highly intertwined structures and interfering processes. Gardner coined the phrase "fiction of separation" to describe this phenomenon in the nonstandard models.

For example, in their POPL'13 work on Views, Dinsdale-Young, Parkinson and colleagues show that a simple abstract version of concurrent separation logic can embed many other techniques for reasoning about concurrency including type systems and even the classic rely-guarantee method, which was invented for the purpose of reasoning about interference. Work of Nanevski and Sergey on their Fine-Grained Concurrent Separation Logic shows how one of the sources of non-modularity in the classic Owicki-Gries approach to concurrency, the treatment of auxiliary variables, can be addressed by a suitable non-standard model of separation. They also obtain great mileage out of a model that interprets the separating conjunction in terms of a composition of histories, thus combining temporal and spatial aspects of reasoning. Finally, Hoare and others have been pursuing a very general theory of "Concurrent Kleene Algebra", which encompasses message passing as well as shared variable concurrency, and its associated program logic is again a very general form of CSL.

Although they technically use simple abstract version of CSL, these works conceptually go well beyond the original because the nonstandard models have meanings so far removed from the standard models. And they represent technically significant advances as well. For instance, the Views work has a new and very flexible proof of soundness, which is needed I think to cover the concurrency in the nonstandard models. There is a lot happening in this space, and I have left out other very good work by Birkedal, Dreyer, Raad, Feng, Shao and others; a number of competing logics are being advanced, and there is just too much good work to mention it all here. It will take some time to see these developments shake out, but already it is clear that the principles of CSL apply much more broadly one could have guessed at the time of mine and Steve's papers.

Second, I would like to mention that a surprising practical development has been the degree of progress in tools for mechanized verification. The first implementation of CSL actually preceded the publication of mine and Steve's papers. Cristiano Calcagno included CSL in his earliest prototypes of Smallfoot, the first separation logic verification tool, around 2002. But, since then, many tools have appeared for verification with CSL and relatives. Examples of the state of the art include the Verifast tool of Jacobs et al. and the implementation in Coq of the aforementioned Fine-grained CSL: in both tools there are examples of mechanized proofs of nontrivial concurrent programs including hand-over-hand locking on linked lists, lock-free queues, and a concurrent spanning tree construction. It is also possible to verify custom synchronisation primitives, such as for locks implemented using compare-and-swap, and then to use the specifications of the primitives in verifications of client code without having to look at the lock internals.

LA: What are some of the current directions of interest, or future problems, for research.

PO: One important direction in pure theory is unification, to bring to a form of local conclusion all of the developments on non-standard models. I sense that there is good and possibly deep theoretical work to be done there.

SB: I agree. And I think the time is ripe for a systematic attempt to develop a denotational framework capable of supporting such a unification. My own recent research into the foundations of weak memory concurrency is an attempt to start in this direction, and seems like a natural generalization from the action trace semantics that we used to formalize CSL. The main idea here is to go from traces (essentially, linearly ordered sets of actions) to a more general partial-order setting. Similar themes are also appearing in work of others, and we are starting to see papers proposing the use of pomsets (my own work, building on early ideas of Vaughan Pratt) and event structures (originally introduced by Winskel) in semantic accounts of weak memory. I think this is exciting, and I believe it would be a valuable service to cast these developments into a uniform framework, and to use such a framework to establish soundness of new CSL-style logics for weak memory and explore the relationships between such logics.

PO: As I said above, there has been tremendous progress in mechanized verification of concurrent programs; but there has been less in automatic program analysis. With program analysis we would like to give programmers feedback without requiring annotations, say by trying to prove specific integrity properties (such as memory safety or race freedom); annotations can help the analysis along, but are not needed to get started, and this greatly eases broad deployment. A number of prototype concurrency analyses based on CSL have been developed, but there has been much more work applying sequential separation logic to program analysis. For example, the Infer program analyser, which is in production at Facebook, uses sequential but not concurrent separation logic. To make advanced program analysis for concurrency which brings value to programmers in the real world is in the main an open problem. And not an easy one.

I would finally like to mention language design. There have been experimental type systems which incorporate ideas from CSL into a programming language, such as the Mezzo language and Asynchronous Liquid Separation Types. Related ideas can be found earlier in Cyclone, and more recently in the ownership typing that happens in the Rust language. It seems as if there is a lot of room for experimentation and innovation in this space.

LA: Peter, Steve, thank you very much for sharing your recollections and knowledge with the theoretical-computer-science community, and congratulations for the 2016 Gödel Prize!

Interview with Rajeev Alur and David Dill 2016 Alonzo Church Award Recipients

Luca Aceto
ICE-TCS, School of Computer Science,
Reykjavik University
luca@ru.is

Rajeev Alur (University of Pennsylvania USA) and David L. Dill (Stanford University, USA) are the recipients of the 2016 Alonzo Church Award for Outstanding Contributions to Logic and Computation for their work on timed automata, a decidable model of real-time systems that combines beautiful, new and deep theory with widespread practical impact.

In particular, Alur and Dill are honoured for their paper "A theory of timed automata", Theoretical Computer Science 126(2):183–235, 1994. Fundamental Study. `http://dx.doi.org/10.1016/0304-3975(94)90010-8`. That paper proposes a variation on classic finite automata to model the behaviour of real-time systems. The resulting notion of timed automaton provides a strikingly simple, and yet powerful, way to annotate state-transition graphs with timing constraints using finitely many real-valued clocks. The fundamental study by Alur and Dill developed the model of timed automata as a formalism for accepting languages of timed words, studied the model from the perspective of formal language theory by considering closure properties and decision problems for the full model and some of its sub-classes, mapped the decidability boundary for the considered decision problems, and introduced a fundamental abstraction, the so-called *region graph*, that has since found application in virtually every decidability result for models of real-time and hybrid systems, and that is at the heart of coarser abstractions that are embodied in the verification engines of several industrial-strength tools for automatic verification of real-time systems.

In the twenty five years since their invention, timed automata have become the standard model for the analysis of continuous-time systems, which underlies hundreds of papers, tens of tools, and several textbooks.

In order to celebrate the award of the Alonzo Church Award to this hugely influential work and to give the theoretical computer science community at large a glimpse of the history of the ideas that led to it, I interviewed Rajeev Alur (abbreviated to RA in what follows) and David Dill (referred to as DD in the text below) via email. I hope that the readers of the Bulletin of the EATCS will enjoy reading the text of the interview and will find it as interesting as I did.

The interview

LA: You are receiving the 2016 Alonzo Church Award for Outstanding Contributions to Logic and Computation for your invention of timed automata, which, as far as I know, was the first decidable formalism for reasoning about the ubiquitous class of real-time systems. Could you briefly describe the history of the ideas that led to the development of the formalism of timed automata, what were the main inspirations and motivations for its invention and how timed automata advanced the state of the art?

DD: The interesting part of the story for me is the inspiration that led to the question in the first place and the initial results. Things worked out so magically that I'm still amazed, so I hope that the story will be interesting to others. I also learned a lot of lessons about research, which I hope I can convey here.

I am not a logician and don't consider myself a theorist. I generally want to do research that has practical impact, but I like being able to apply theory to help with that. And, in this case, I ended up creating some theory as part of that process.

My PhD research under Ed Clarke at CMU was on using finite automata for formal verification of speed independent circuits. I started working on them using CTL model checking, and then decided during my thesis work to abandon model checking and used finite automata (I am grateful that Ed accepted this change in direction, since his primary research agenda was CTL model checking). Speed-independent circuits are digital circuits with no clocks, which are supposed to meet their specifications for all possible gate delays. Ed had recently co-invented CTL model checking and was exploring speed-independent circuits as an application, because speed-independent circuits are among the simplest concurrent systems with bugs. But speed independence is a very conservative model, because engineers often know some constraints on delays, even if they can't specify exact values for the delays. A circuit designer (Prof. Chuck Seitz of CalTech) had some ways of designing speed-independent circuits that relied on timing constraints without precisely specified delays, and asked me whether I could prove such circuits correct.

I looked at the literature, and there were a number of people who had used finite automata with a special alphabet symbol representing clock ticks, and they would simply count the clock ticks using the states of the automaton (what was later called a discrete time model). But I wasn't comfortable with that, because asynchronous circuits don't have clocks! I felt that events in circuits happened in continuous time, which might be different from discrete time – but I didn't know.

While I was working on my PhD, I sat down to try to work out a method to verify timed asynchronous circuits. The research was painless compared to a lot of my other PhD work. I imagined that each gate had a timer that was set to a real value between constant delay bounds, and that these timers decreased continually with time until the reached 0, at which point the gate output would change. Very quickly, I came up with an algorithm that kept track of the set of all possible timer values for various discrete states, and, magically, the analysis of the regions could be easily solved using the all-pairs shortest paths problem. I think I just got lucky and happened to think about it the right way, so it was easy to do. (Later, I learned that Vaughan Pratt had observed that certain systems of linear inequalities could be solved using shortest paths, and that a similar algorithm was used in a kind of timed Petri nets – but I didn't know that at the time).

I left this method out of my PhD thesis, because my thesis seemed coherent and the timed model didn't seem to mesh with the other material. Then, in my first few years at Stanford, I pulled it out again and tried to prove that it was actually sound. Getting the details right was harder than working out the original idea. It took several weeks. The paper eventually appeared in CAV '89.

Verification with linear time (as opposed to branching time) models seemed beautiful to me because all the problems reduced to standard closure properties and decision properties that had been solved for finite automata: Closure under intersection, complementation, and projection, and the decidability of language emptiness. Implicit in my CAV paper were closure under intersection, projection, and the decidability of emptiness for timed regular languages, but I couldn't prove closure under complementation. I felt very strongly that timed regular languages were closed under complementation and would work the same way as conventional finite automata.

Then Rajeev Alur walked into my office. He was a second year student who was ready to do some research, and his research advisor, Zohar Manna, was out of the country for a while. I explained timed automata to Rajeev and asked whether he could resolve the question of whether they were closed under complementation. Rajeev very quickly came up with a nicer definition of timed automata, with clocks that counted up and predicates, and an "untiming" construction for deciding language emptiness that had lower complexity than mine. I remember it took him several months to prove that, in fact, timed automata were not closed under complementation and that, surprisingly, the universality problem was undecidable. He proved several other amazing results, and then we wrote the "Timed Automata" paper. The paper was rejected twice from conferences with pretty harsh reviews (I think FOCS was one conference — I don't remember the other one) and was eventually accepted at ICALP.

I'm not sure why it was so poorly received. I don't remember changing it much when we resubmitted it. I think there is a problem when you formulate a new problem and solve it, but reviewers don't understand why the problem is important. If the solution is a bit difficult, they look for reasons to reject the paper. If you attack a well-known problem so everyone knows why you want to solve it, it's sometimes easier to sell the paper (but maybe it won't have as much impact). Or maybe we just had some bad luck — everyone gets bad reviews, and I've unfortunately written some bad reviews myself.

RA: I joined Stanford University in 1987 as a graduate student primarily interested in logic and theory of computation. Topics such as model checking, temporal logics, and automata over infinite strings were hot topics then, and extending these formalisms to reasoning about real-time systems was a natural research direction.

For me, the key starting point was Dave's wonderful paper titled "Timing assumptions and verification of finite-state concurrent systems" that appeared in CAV 1989. Dave has already explained how he arrived at the idea of difference bounds matrices that appeared originally in his paper, and I will explain two new ideas in our follow-up work aimed at generalizing the results. Dave's original paper modeled timing constraints by introducing timers each of which could be set to a value chosen nondeterministically from a bounded interval, decreased with time, and triggered an event upon hitting 0. We changed the model by introducing clock variables each of which could be reset to 0, increased with time, and could be tested on guards of transitions. In retrospect, this led to a simpler model, and more importantly, led naturally to many generalizations such as hybrid automata. The second idea was the definition of region equivalence that defines a finite quotient over the infinite set of clock values. This proved to be a versatile tool and forms the basis of many decidability results, and in particular, algorithms for model checking of branching-time temporal logics.

LA: According to Google Scholar, the journal paper for which you receive the Alonzo Church Award, which was published in 1994, has so far received over 6,500 citations, whereas the ICALP 1990 on which it was based has been cited over 1,250 times. A Google Scholar query also reveals that at least 135 publications citing your landmark TCS paper have more than 135 citations themselves. Moreover, the most successful and widely used tools for modelling and verification of real-time systems are based on timed automata. When did it dawn on you that you had succeeded in finding a very good model for real-time systems and one that would have a lot of impact? Did you imagine that your model would generate such a large amount of follow-up work?

DD: It will be really interesting to hear what Rajeev says about this.

At the time, I just felt happy that we had a nice theory. I was looking around for other uses besides asynchronous circuits to which timed automata would be applicable, and, based on the name, "real-time systems" seemed like a good candidate. But I didn't know much about the practical side of that area, and timed automata didn't seem to be directly applicable to many of the problems they worried about (there were a **lot** of papers on scheduling!).

It took a very long time before I was confident that timed automata were useful for anything, including real-time systems. So, for me, it was based on feedback from other people. I got some grant money to work on the problem, which was a good sign. People from control theory and real-time systems invited me to come and talk to them about it. It may be that timed model checking (which is very closely related) has had more practical impact than timed automata themselves. Other people built tools that were better than anything I created.

After a few years, the area got too hard for me. Others, especially Rajeev, were doing such a good job that I didn't feel that my efforts were needed. So, I moved on to new topics in formal verification and asynchronous circuit design. Now I only have a fuzzy idea about the impact of timed automata, embarrassingly.

RA: The initial reception to our work was not enthusiastic. In fact, the paper was rejected twice, once from FOCS and once from STACS, before it was accepted in ICALP 1990. There was also a vigorous argument advocated by a number of prominent researchers that modeling time as a discrete and bounded counter was sufficient. By mid 1990s though the model started gaining acceptance: in theory conferences such as CONCUR, ICALP, and LICS a steady stream of papers studying complexity of various decision problems on timed automata and extensions started, and a number of implementations such as KRONOS and UPPAAL were developed. However, I could have never imagined so much follow-up work, and I feel very grateful to all the researchers who have contributed to this effort.

LA: What is the result of yours on timed automata you are most proud of? And what are your favourite results amongst those achieved by others on timed automata?

DD: I'm most proud of coming up with the question and the initial (not so hard) results. To me, the question seemed so completely obvious that I couldn't believe it hadn't been answered. I contributed to some of the later results, but Rajeev took off like a rocket and I couldn't keep up with him. At some point, there was so much work in the area that I didn't feel I had much to add. I know there are a lot of amazing results that I haven't studied.

I like the fact that timed automata formed a paradigm that others followed with hybrid automata and so on. The idea of extending finite automata with other gadgets, like clocks, and analyzing the resulting state space seems to have influenced people as much as the results on timed automata.

RA: There are lots of strong and unexpected results that I like and it is hard to choose. But since you asked, let me point out two which are closely related to our original paper. The paper by Henzinger et al. titled "Symbolic model checking of real-time systems" (LICS 1992) introduced the idea of associating invariants with states. I think this is a much cleaner way of expressing upper bounds on delays and, in fact, is now part of the standard definition of timed automata. In our original paper, we had proved undecidability of timed language equivalence. Cerans showed in 1992 decidability of timed bisimulation which I did not expect and involves a clever use of region equivalence on product of two timed automata.

LA: Twenty five years have passed since the invention of timed automata and the literature on variations on timed automata as well as logics and analysis techniques for them is huge. Do you expect any further development related to theory and application of timed automata in the coming years? What advice would you give to a PhD student who is interested in working on topics related to timed automata today?

DD: I'm sorry, but I haven't actively pursued the area and I just don't know. If I were giving advice to a younger version of me, knowing my strengths and weaknesses, I would actually advise that person to go and find a new problem that hasn't been worked on much. Maybe you'll start a new field, and you'll get to solve the first problems in the area before it gets too hard. What has worked for me was looking at an application problem, trying to find a clean way to formalize it, and then looking at the problems stemming from that formalization. It's an amazing fact that there are fundamental questions that haven't been addressed at all. Starting with a practical problem and then thinking about what theory would be helpful to solve it is a good way to come up with those questions. Also, watch for cases where existing theory doesn't exactly fit. Instead of trying to pound nails with a wrench, imagine a more appropriate, but simple, theory and invent that.

RA: I feel that theoretical problems as well as verification tools have been extensively investigated. Given that, my advice would be to first focus on an application. When one tries to apply an existing technique/tool to a real-world problem, there is invariably a mismatch and that insight can suggest a research idea. The emerging area of cyber-physical systems is a rich source of applications. For instance, formal modeling and analysis of medical devices can be a fruitful direction.

LA: You have both been involved in inter-disciplinary research with colleagues from biology and control theory. What general lessons have you learned from those experiences? What advice would you give a young researcher who'd like to pursue that kind of research?

DD: On reflection, my research in formal verification was not as collaborative as it could have been. But in computational biology, if you don't have a wet lab, you **really** have to collaborate with other people! Collaboration can be rewarding but also frustrating. My advice would be: learn as much about the other area as you can, so you can at least talk the same language as your collaborators. And prepare to invest lots of time communicating and understanding what motivates them – and make sure they're willing to do the same, otherwise, it's a collaboration that is not going to work out. What keeps you working together is **mutual** benefit.

Most collaborations don't work out. If you have a good collaborator, be grateful. And, sometimes, you may want to choose among research directions based on which ones have the best collaborators.

LA: David, Rajeev, many thanks for taking the time to share your views with the members of the TCS community and congratulations for receiving the first Alonzo Church Award!

E u r o p e a n

A s s o c i a t i o n   f o r

T h e o r e t i c a l

C o m p u t e r

S c i e n c e

E A T C S

155

EATCS

HISTORY AND ORGANIZATION

EATCS is an international organization founded in 1972. Its aim is to facilitate the exchange of ideas and results among theoretical computer scientists as well as to stimulate cooperation between the theoretical and the practical community in computer science.

Its activities are coordinated by the Council of EATCS, which elects a President, Vice Presidents, and a Treasurer. Policy guidelines are determined by the Council and the General Assembly of EATCS. This assembly is scheduled to take place during the annual **I**nternational **C**olloquium on **A**utomata, **L**anguages and **P**rogramming (ICALP), the conference of EATCS.

MAJOR ACTIVITIES OF EATCS

- Organization of ICALP;
- Publication of the "Bulletin of the EATCS;"
- Award of research and academic career prizes, including the EATCS Award, the Gödel Prize (with SIGACT), the Presburger Award, the EATCS Distinguished Dissertation Award, the Nerode Prize (joint with IPEC) and best papers awards at several top conferences;
- Active involvement in publications generally within theoretical computer science.

Other activities of EATCS include the sponsorship or the cooperation in the organization of various more specialized meetings in theoretical computer science. Among such meetings are: CIAC (Conference of Algorithms and Complexity), CiE (Conference of Computer Science Models of Computation in Context), DISC (International Symposium on Distributed Computing), DLT (International Conference on Developments in Language Theory), ESA (European Symposium on Algorithms), ETAPS (The European Joint Conferences on Theory and Practice of Software), LICS (Logic in Computer Science), MFCS (Mathematical Foundations of Computer Science), WADS (Algorithms and Data Structures Symposium), WoLLIC (Workshop on Logic, Language, Information and Computation), WORDS (International Conference on Words).

Benefits offered by EATCS include:
- Subscription to the "Bulletin of the EATCS;"
- Access to the Springer Reading Room;
- Reduced registration fees at various conferences;
- Reciprocity agreements with other organizations;
- 25% discount when purchasing ICALP proceedings;
- 25% discount in purchasing books from "EATCS Monographs" and "EATCS Texts;"
- Discount (about 70%) per individual annual subscription to "Theoretical Computer Science;"
- Discount (about 70%) per individual annual subscription to "Fundamenta Informaticae."

Benefits offered by EATCS to Young Researchers also include:
- Database for Phd/MSc thesis
- Job search/announcements at Young Researchers area

(1) THE ICALP CONFERENCE

ICALP is an international conference covering all aspects of theoretical computer science and now customarily taking place during the second or third week of July. Typical topics discussed during recent ICALP conferences are: computability, automata theory, formal language theory, analysis of algorithms, computational complexity, mathematical aspects of programming language definition, logic and semantics of programming languages, foundations of logic programming, theorem proving, software specification, computational geometry, data types and data structures, theory of data bases and knowledge based systems, data security, cryptography, VLSI structures, parallel and distributed computing, models of concurrency and robotics.

Sites of ICALP meetings:

- Paris, France 1972
- Saarbrücken, Germany 1974
- Edinburgh, UK 1976
- Turku, Finland 1977
- Udine, Italy 1978
- Graz, Austria 1979
- Noordwijkerhout, The Netherlands 1980
- Haifa, Israel 1981
- Aarhus, Denmark 1982
- Barcelona, Spain 1983
- Antwerp, Belgium 1984
- Nafplion, Greece 1985
- Rennes, France 1986
- Karlsruhe, Germany 1987
- Tampere, Finland 1988
- Stresa, Italy 1989
- Warwick, UK 1990
- Madrid, Spain 1991
- Wien, Austria 1992
- Lund, Sweden 1993
- Jerusalem, Israel 1994
- Szeged, Hungary 1995
- Paderborn, Germany 1996
- Bologne, Italy 1997
- Aalborg, Denmark 1998
- Prague, Czech Republic 1999
- Genève, Switzerland 2000
- Heraklion, Greece 2001
- Malaga, Spain 2002
- Eindhoven, The Netherlands 2003
- Turku, Finland 2004
- Lisabon, Portugal 2005
- Venezia, Italy 2006
- Wrocław, Poland 2007
- Reykjavik, Iceland 2008
- Rhodes, Greece 2009
- Bordeaux, France 2010
- Zürich, Switzerland 2011
- Warwick, UK 2012
- Riga, Latvia 2013
- Copenhagen, Denmark 2014
- Kyoto, Japan 2015

(2) THE BULLETIN OF THE EATCS

Three issues of the Bulletin are published annually, in February, June and October respectively. The Bulletin is a medium for *rapid* publication and wide distribution of material such as:

- EATCS matters;
- Technical contributions;
- Columns;
- Surveys and tutorials;
- Reports on conferences;
- Information about the current ICALP;
- Reports on computer science departments and institutes;
- Open problems and solutions;
- Abstracts of Ph.D. theses;
- Entertainments and pictures related to computer science.

Contributions to any of the above areas are solicited, in electronic form only according to formats, deadlines and submissions procedures illustrated at `http://www.eatcs.org/bulletin`. Questions and proposals can be addressed to the Editor by email at `bulletin@eatcs.org`.

(3) OTHER PUBLICATIONS

EATCS has played a major role in establishing what today are some of the most prestigious publication within theoretical computer science.

These include the *EATCS Texts* and the *EATCS Monographs* published by Springer-Verlag and launched during ICALP in 1984. The Springer series include *monographs* covering all areas of theoretical computer science, and aimed at the research community and graduate students, as well as *texts* intended mostly for the graduate level, where an undergraduate background in computer science is typically assumed.

Updated information about the series can be obtained from the publisher.

The editors of the EATCS Monographs and Texts are now M. Henzinger (Vienna), J. Hromkovič (Zürich), M. Nielsen (Aarhus), G. Rozenberg (Leiden), A. Salomaa (Turku). Potential authors should contact one of the editors.

EATCS members can purchase books from the series with 25% discount. Order should be sent to:

Prof.Dr. G. Rozenberg, LIACS, University of Leiden,
P.O. Box 9512, 2300 RA Leiden, The Netherlands

who acknowledges EATCS membership and forwards the order to Springer-Verlag.

The journal *Theoretical Computer Science*, founded in 1975 on the initiative of EATCS, is published by Elsevier Science Publishers. Its contents are mathematical and abstract in spirit, but it derives its motivation from practical and everyday computation. Its aim is to understand the nature of computation and, as a consequence of this understanding, provide more efficient methodologies.

The Editor-in-Chief of the journal currently are D. Sannella (Edinburgh), L. Kari and P.G. Spirakis (Patras).

ADDITIONAL EATCS INFORMATION

For further information please visit `http://www.eatcs.org`, or contact the President of EATCS:

Prof. Dr. Luca Aceto,
School of Computer Science
Reykjavik University
Menntavegur 1 IS-101 Reykjavik, Iceland
Email: `president@eatcs.org`

EATCS MEMBERSHIP

DUES

The dues are €30 for a period of one year (two years for students / Young Researchers). Young Researchers, after paying, have to contact `secretary@eatcs.org`, in order to get additional years. A new membership starts upon registration of the payment. Memberships can always be prolonged for one or more years.

In order to encourage double registration, we are offering a discount for SIGACT members, who can join EATCS for €25 per year. We also offer a five-euro discount on the EATCS membership fee to those who register both to the EATCS and to one of its chapters. Additional €25 fee is required for ensuring the *air mail* delivery of the EATCS Bulletin outside Europe.

HOW TO JOIN EATCS

You are strongly encouraged to join (or prolong your membership) directly from the EATCS website `www.eatcs.org`, where you will find an online registration form and the possibility of secure online payment. Alternatively, contact the Secretary Office of EATCS:

Mrs. Efi Chita,
Computer Technology Institute & Press (CTI)
1 N. Kazantzaki Str., University of Patras campus,
26504, Rio, Greece
Email: `secretary@eatcs.org`,
Tel: +30 2610 960333, Fax: +30 2610 960490

If you are an EATCS member and you wish to prolong your membership or renew the subscription you have to use the Renew Subscription form. The dues can be paid (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.

BNP Paribas Fortis Bank, Driekoningenstraat 122, 2600 Berchem - Antwerpen, Belgium
Account number: 220–0596350–30–01130
IBAN code: BE 15 2200 5963 5030, SWIFT code: GEBABE BB 18A

For adittional information please contact the Secretary of EATCS:

Prof. Ioannis Chatzigiannakis,
via Ariosto 25, floor II, room B214,
Sapienza University of Rome,
Rome 00185, Italy
Email: `secretary@eatcs.org`,
Tel: +39 677274073, Fax: +39 0677274002