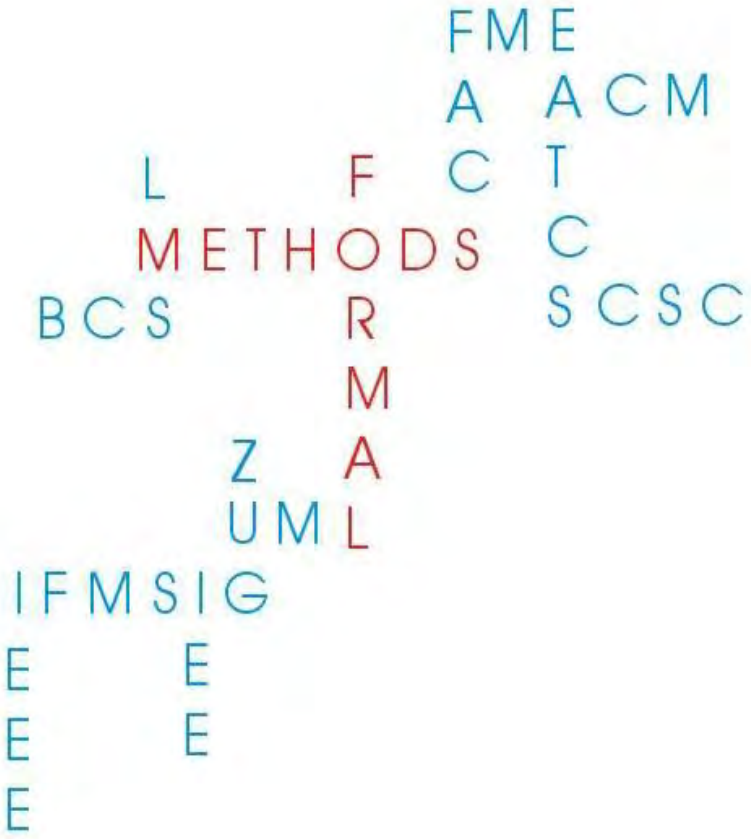
Issue 2022-1
January 2022

# FACS

# A C T S




bcs The Chartered Institute for IT
Formal Aspects of Computing Science Specialist Group

**The Newsletter of the Formal Aspects of Computing Science (FACS) Specialist Group**

ISSN 0950-1231

# About FACS FACTS

FACS FACTS (ISSN: 0950-1231) is the newsletter of the BCS Specialist Group on Formal Aspects of Computing Science (FACS). FACS FACTS is distributed in electronic form to all FACS members.

Submissions to FACS FACTS are always welcome. Please visit the newsletter area of the BCS FACS website for further details at:

https://www.bcs.org/membership/member-communities/facs-formal-aspects-of-computing-science-group/newsletters/

Back issues of FACS FACTS are available for download from:

https://www.bcs.org/membership/member-communities/facs-formal-aspects-of-computing-science-group/newsletters/back-issues-of-facs-facts/

# The FACS FACTS Team

**Newsletter Editors:**

Tim Denvir timdenvir@bcs.org
Brian Monahan brianqmonahan@googlemail.com

**Editorial Team:**

Jonathan Bowen, John Cooke, Tim Denvir, Brian Monahan, Margaret West.

**Contributors to this issue:**

Richard Bornat, Jonathan Bowen, Tim Denvir, Renaud Di Francesco, Egon Börger, Rainer Glaschick, Keith Lines, Brian Monahan, Rajagopal Nagarajan, Peter Sewell, John Tucker, Margaret West, Glynn Winskel

# BCS-FACS websites

**BCS:** http://www.bcs-facs.org
**LinkedIn:** https://www.linkedin.com/groups/2427579/
**Facebook:** http://www.facebook.com/pages/BCS-FACS/120243984688255
**Wikipedia:** http://en.wikipedia.org/wiki/BCS-FACS

If you have any questions about BCS-FACS, please send these to Jonathan Bowen at jonathan.bowen@lsbu.ac.uk.

# Editorial

Dear readers,

Welcome to issue 2022-1 of the *FACS FACTS* newsletter. This being our first issue after the 2021 AGM, we begin with our chairman, Jonathan Bowen's report for the last year, strait-jacketed into the required BCS format.

Our journal, *Formal Aspects of Computing*, published by Springer for the last 32 years, is from this year onwards going to be published by the ACM. The press announcement follows Jonathan Bowen's report.

This newsletter, including its contributions, is edited, composed, and prepared by well-informed volunteers with formal methods and computing science backgrounds. Unlike the journal, we encourage a less traditional style of article that invites open discussion and reflection instead of review-style commentary. Even the best traditional papers have some of this style of sharing thoughts, rather than making proclamations. A great example of this, among others, can be seen in the works of the late Robin Milner, we feel.

This policy naturally means that, from time to time, we receive articles for which we cannot conduct as rigorous a review as we might otherwise desire but for which we nonetheless see there is sufficiently interesting content. A case in point in this newsletter is Richard Bornat's report of his own seminar on aspects of Quantum Computation. He shares his thought processes as he mulls over a number of questions, and invites conversation, unlike a traditional paper, which might assert a claim or position. We therefore invite, indeed urge, responses to this and other contributions: we greatly hope that conversation will ensue!

There is, therefore, an element of trust between us, the newsletter team, and readers, that published articles are worth reading and at least have something interesting to say. The absence of review means that we inevitably *must* rely on you, the vigilant reader, to actively help spot potentially controversial or debatable issues that arise – and then for readers to provide considered responses, as occasion demands.

We consider all contributions as being potentially suitable for publication—and, as ever, we endeavour to do our best in exercising our judgement in that regard. We shall continue to support a broad range of document formats for contributions – currently, basic text, Word/ODF, LaTeX, and PDF – however, we may have to start limiting the length of contributions, particularly for those in LaTeX and PDF formats, to around 40 pages or so. We are also considering some alternatives to make the production process less time consuming.

Besides the chairman's report and the ACM announcement, in this issue there are reports of various FACS and related seminars, two book reviews, a further report from

the *History of Computing* collection at Swansea, and a feature on the *Turing Tradition at the Logic School of Münster*. A table of contents (with links) follows on the next page.

We hope you enjoy *FACS FACTS* issue 2022-1.

Tim Denvir
Brian Monahan

PS: At the recent Annual Landin Lecture this year, Tim set a one-question Christmas quiz:

> *A little scurrilous. But this is the last meeting of FACS this year. 2021 is a very special year, the like of which will not be seen again during our lifetimes or the lifetimes of our descendants, nor the lifetimes of our ancestors for many generations. The last time there was a year like 2021 was 1763, and the next will not be for over four centuries in 2491. So, a one-question Christmas quiz: what is special about 2021 and those other years?*
>
> (Answer at foot of page 48)

# Table of Contents

# *BCS-FACS Specialist Group 2021 Chair's Report*

| | |
|---|---|
| **Member Group Name:** | FACS Specialist Group |
| **Year:** | 2021 |
| **Report By:** | Jonathan Bowen |

| | |
|---|---|
| **Group Chair:** | Jonathan Bowen |
| **Group Treasurer:** | John Cooke |
| **Group Secretary:** | Roger Carsley |
| **Group Inclusion Officer:** | Margaret West |
| **Other Committee Members:** | Ana Cavalcanti (FME Liaison), Tim Denvir (*FACS FACTS* newsletter co-editor), Brijesh Dongol (Refinement Workshop Liaison), Rob Hierons (LMS Liaison), Keith Lines (Government and Standards Liaison), Brian Monahan (*FACS FACTS* newsletter co-editor) |

## Successes

| **Success** | **Additional Comments** |
|---|---|
| 1. Continued evening seminars online, with recordings on YouTube | The BCS Zoom facilities and recording transfer to YouTube have widened access to FACS seminars. Thank you to Keith Lines, Ana Cavalcanti, and Rob Hierons, for help with organising 2021 seminars. |
| 2. Publication of *FACS FACTS* newsletters | We now aim for two major newsletters each year, published online in PDF format. Thank you to Tim Denvir and Brian Monahan for sterling work in editing the 2021 newsletters. |
| 3. Move to hybrid (online and in-person) evening seminars at the end of 2021 | The last two 2021 seminars at the London Mathematical Society (November) and our major Peter Landin Semantics Seminar at the BCS London office (December) are being delivered in hybrid format. |

| Planned Activity | Additional Comments |
|---|---|
| 1. Continued hybrid evening seminars | The BCS facilities for hybrid talks should enable this mode of delivery. |
| 2. At least two *FACS FACTS* newsletters | We aim for January and probably July as months of publication. |
| 3. Collaboration with Formal Methods Europe (FME) and London Mathematical Society (LMS) organisations | Note that our LMS Liaison Officer plans to retire at the end of 2021, so we need a new volunteer. |

| Impediment | Description |
|---|---|
| 1. Lack of volunteers to organise evening seminars | This has mainly fallen to the Chair and Keith Lines in 2021 but the position of Seminar Organiser is still vacant. It is an opportunity to invite speakers you would like to meet and hear. Volunteers are welcome! |
| 2. Our LMS Liaison Officer is retiring (see above) | Thank you to Rob Hierons for organising the 2021 FACS/LMS seminar. We now need a new and keen person in this post. It is an opportunity to be involved with both FACS and the LMS. The main role is organizing an annual joint seminar each November. We believe that we have a volunteer! |
| 3. Covid restrictions | The lack of physical meetings has impeded networking of FACS members. Hopefully this will be alleviated with hybrid events. |

## Additional Facts and Figures

We aim for at least two *FACS FACTS* newsletters per year (with two in 2021, in February and July). We also aim for around six evening seminars per year (with seven in 2021).

## Further Comments

The Covid pandemic has continued to affect activities significantly. We have had no physical meetings during most of the pandemic, all being online until the start of hybrid events from November 2021. We organized the following evening seminars during 2021:

1 *NPL's Experience with Formal Aspects*, by **Keith Lines**, National Physical Laboratory, 6 April.

2 *New Ways of Using Formal Models in Industry*, by **Michael Leuschel**, Heinrich-Heine-Universität Düsseldorf, Germany, 6 May. In association with Formal Methods Europe (FME).

3 *Dimensionally Correct by Construction: Type systems for programs*, by **Conor McBride** and **Fredrik Nordvall Forsberg**, University of Strathclyde, 15 June.

4 *Matrices of Sets*, by **Renaud Di Francesco**, Sony Europe BV, 23 September.

5 *Formal Modelling, Programming and Verification of Quantum Systems*, by **Rajagopal Nagarajan** and **Richard Bornat**, Middlesex University London, 19 October.

6 *Underpinning Mainstream Engineering with Mathematical Semantics*, by **Peter Sewell**, University of Cambridge, 18 November. In association with the London Mathematical Society (LMS), at the LMS headquarters.

7 *Making Concurrency Functional*, by **Glynn Winskel**, University of Cambridge, 17 December. The annual Peter Landin Semantics Seminar, in association with the FACS AGM, at the BCS London office.

Thank you to all the FACS committee members for performing their various roles, as detailed above. In addition, John Cooke attended an online BCS Member Group Working Group meeting and Margaret West is always helpful with useful comments! New committee members are very welcome, especially if interested in organising seminars and LMS liaison, as previously mentioned.

# *News Release: ACM to publish BCS FACS Journal*

Association for Computing Machinery

Advancing Computing as a Science & Profession

The Chartered Institute for IT

**NEWS RELEASE**

**Contact:** Jim Ormond
212-626-0505
ormond@hq.acm.org

Amanda Matheson
07793 837385
amanda.matheson@bcs.uk

**ACM to Publish BCS, The Chartered Institute for IT's *Formal Aspects of Computing* Journal**

***Influential Publication Will Become Gold Open Access Effective January 2022***

**New York, NY, August 4, 2021**—ACM, the Association for Computing Machinery, and BCS, The Chartered Institute for IT (BCS), are pleased to announce that they have entered into a long-term co-publishing Agreement for the influential BCS journal *Formal Aspects of Computing*. Effective January 1, 2022, the Journal will be published by ACM as a fully Gold Open Access journal, with all published articles in the Journal, including the complete archive of previously published articles dating back to 1989, being Open Access in the ACM Digital Library.

Currently published by Springer Nature for BCS, *Formal Aspects of Computing* welcomes new theoretical contributions where they are motivated by potential application, and applications of formalisms, where they show something novel about their approach or application.

The scope of *Formal Aspects of Computing* includes well-founded notations for the description of systems, verifiable design methods, elucidation of fundamental computational concepts, approaches to fault-tolerant design, theorem-proving support, state-exploration tools, formal underpinning of widely used notations and methods, formal approaches to requirements analysis, and history of formal methods.

An important goal of the new ACM/BCS partnership is to broaden both the editorial scope of the journal and the journal's international reach.

"*Formal Aspects of Computing* recently entered its fourth decade, and the relevance of the tools and theories we explore continues to grow across all areas of computing," explained Editor-in-Chief James Woodcock of the University of York. "At the same time, we believe the journal will benefit from having a more international focus as an ACM publication. As the largest association of computing professionals in the world, ACM offers the perfect platform to attract a much larger pool of researchers to contribute to and read the journal."

"As professional computing societies, ACM and BCS have a shared mission to provide the highest quality resources for the career development of our members and to advance research and education to benefit the public," added Ian Borthwick, Head of Publishing for BCS. "The improved reach and

international expansion of *Formal Aspects of Computing* will help accelerate the growth and impact of the Journal, as well as broaden the diversity of article submissions. The journal will be a more accessible resource for BCS members and the broader computing community to read and learn from, while raising the profile of researchers who publish in it. We're especially pleased that this new arrangement will allow for Gold Open Access, which is a growing trend in research publishing. We are pleased to invite all ACM members and stakeholders to explore the *Formal Aspects of Computing* backfile. Outstanding research has been published in this journal, which we are happy to bring to the attention of a wider audience."

"BCS have done an exceptional job of making *Formal Aspects of Computing* one of the leading journals in this area and it fills an important niche within ACM's growing family of journals," added Scott Delman, ACM's Director of Publications. "James Woodcock will remain the Editor-in-Chief of the Journal and ACM will work closely with Jim and our global volunteer community of respected computer scientists to add more geographic diversity and subject matter expertise to the editorial board. Transitioning the Journal to Gold Open Access as from 2022, leveraging both the ACM OPEN model and the option for authors to open their articles in the Journal by paying an affordable Article Processing Charge (APC), reflects ACM's strong commitment to making computing research accessible to the broadest possible audience of researchers, educators, students, and practitioners. ACM looks forward to being part of the next chapter of this respected publication and further collaborations with BCS."

ACM publishes more than 60 scholarly peer-reviewed journals in dozens of computing and information technology disciplines. Available in print and online, ACM's high-impact, peer-reviewed journals constitute a vast and comprehensive archive of computing innovation, covering emerging and established computing research for both practical and theoretical applications. ACM journal editors are thought leaders in their fields, and ACM's emphasis on rapid publication ensures minimal delay in communicating exciting new ideas and discoveries.

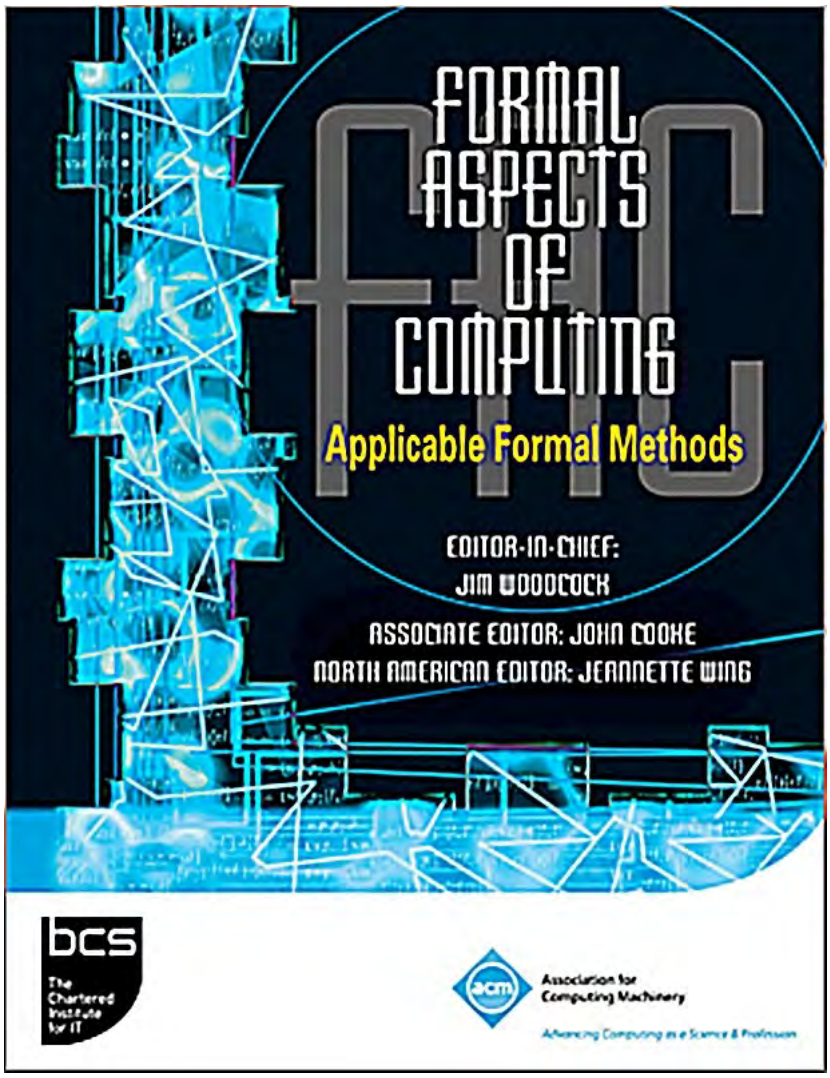
ACM's transition to Open Access is driven by the rapid growth of the ACM OPEN model. Since its launch in late January 2020, more than 140 institutions worldwide have committed to the ACM OPEN license model. The newly signed agreement with BCS continues ACM's collaborative efforts toward becoming a sustainable, fully Open Access research publisher for the computing community.

**About ACM**

ACM, the Association for Computing Machinery, is the world's largest educational and scientific computing society, uniting educators, researchers, and professionals to inspire dialogue, share resources and address the field's challenges. ACM strengthens the computing profession's collective voice through strong leadership, promotion of the highest standards, and recognition of technical excellence. ACM supports the professional growth of its members by providing opportunities for life-long learning, career development, and professional networking.

**About the BCS, The Chartered Institute for IT**

BCS is the UK's Chartered Institute for IT. The purpose of BCS as defined by its Royal Charter is to promote and advance the education and practice of computing for the benefit of the public. We bring together industry, academics, practitioners, and government to share knowledge, promote new thinking, inform the design of new curricula, shape public policy and inform the public.


FORMAL ASPECTS OF COMPUTING
Applicable Formal Methods
EDITOR-IN-CHIEF:
JIM WOODCOCK
ASSOCIATE EDITOR: JOHN COOKE
NORTH AMERICAN EDITOR: JEANNETTE WING
bcs
The Chartered Institute for IT
Association for Computing Machinery

# *Matrices of Sets*

Renaud Di Francesco, Sony Europe

Webinar presented: 23/09/2021

https://www.youtube.com/watch?v=Jb7g8-QA2k0

Reported by: Keith Lines, NPL

## Introduction

On 23rd September FACS hosted, via a webinar, the first presentation for the BCS by Dr Renaud Di Francesco: Director, Europe Technology Standards Office, Sony Europe.

The subject was matrices of sets, about which he has also presented this year at the National Physical Laboratory UK, Université Le Havre Normandie, Université Gustave Eiffel and Coventry University. Matrices of sets [1] is an interesting concept that is being applied to use-cases such as freight logistics [2,3].

## Summary

The classic legend of the grains of rice on a chessboard [4] was used to explain how a matrix of sets can provide a richer data model, than a matrix of numbers alone. Each grain of rice can be thought of as an element of a set. For “grain of rice” it could, for example, be possible to substitute “unique serial number of an artefact”.

Some historical background was presented, such as Cayley's introduction of matrices in the late 1850s [5]. The work of Cantor [6] and Hilbert [7], amongst others, was also touched upon.

The ability to perform operations distinguishes a matrix from what would otherwise just be a straightforward data structure. The matrix of sets equivalent of multiplication of purely numeric matrices was described, along with matrix of sets equivalents of eigenvectors and eigenvalues. Definitions of the addition and difference operators are provided in [1].

Theorems concerning eigenvalues and triangular matrices were presented, along with the proofs. There was also a description of a *Matryoshka* (or Russian Doll) property that simplifies multiplication of matrices of sets. I.e. for two matrices that are being multiplied, *A* and *B*, for row *i* from *A* and its equivalent column *j* from B:

$$A_{i,k}A_{i,k+1} \text{ and } B_{k,j}B_{k+1,j} \text{ where } k = 1 \text{ to } M,$$

where *M* is the number of rows in *A* and columns in *B*

There was also a description of using matrices of sets to generate polynomials.

A proposal was made that matrices of sets can assist with the anonymisation of data.

Attention then turned to use-cases. As noted above, and in [2, 3], freight logistics has proved to be a fruitful area of application. Collaborative filtering in recommendation systems could also benefit from the rich data representation provided by matrices of sets.

Finally, some next steps were proposed:

- Apply the Gram-Schmidt process to matrices of sets
- The use of Loeb's sets with a negative number of elements [8]

## Questions and Answers

Topics covered included:

- Standardisation has been an important part of Renaud's career. Is there a role for formal aspects in developing standards? Yes, but some strong use-cases would be required.
- What software tool support is there for matrices of sets? Development of such tools may be an interesting project for an expert in Mathematica or MATLAB. Colleagues at the Université Le Havre Normandie have written code to implement matrices of sets. It may prove difficult to represent matrices of sets using MATLAB.
- What about the use of multi-dimensional arrays compared to matrices of sets?

Keith Lines, Data Science Department,
National Physical Laboratory UK, December 2021


*View of the talk on Zoom (screenshot by Jonathan Bowen).*

# References


1 Di Francesco R., Matrices of Sets, complete tutorial with use cases (July 2021)
Retrieved 16/12/2021 from Research Gate
https://www.researchgate.net/publication/353246623_Matrices_of_Sets_complete_tutorial_with_use_cases

2 Di Francesco R., Containers transport and logistics models with Matrices of Sets: - enabling digital efficiency gains for freight transport & logistics (July 2021)
Retrieved 16/12/2021 from Research Gate
https://www.researchgate.net/publication/352904491_Containers_transport_and_logistics_models_with_Matrices_of_Sets_-enabling_digital_efficiency_gains_for_freight_transport_logistics

3 Di Francesco R., Maritime Economics Computable Models using Matrices of Sets: study cases of 1) economics of routing for multimodal transport, 2) expression of preference across heterogeneous dynamic baskets Applications illustrating the efficiency of economic modelling with Matrices of Sets (July 2021)
Retrieved 16/12/2021 from Research Gate
https://www.researchgate.net/publication/352904692_Maritime_Economics_Computable_Models_using_Matrices_of_Sets_study_cases_of_1_economics_of_routing_for_multimodal_transport_2_expression_of_preference_across_heterogeneous_dynamic_baskets_Applications_

4 The Rice and Chessboard Legend
Retrieved 16/12/2021 from the Institute of Mathematics and its Applications
https://www.mathscareers.org.uk/the-rice-and-chessboard-legend

5 Cayley A., A memoir on the theory of matrices (January 1858)
Retrieved 16/12/2021 from the Royal Society
https://royalsocietypublishing.org/doi/10.1098/rstl.1858.0002

6 Cantor, G., Contributions to the Founding of the Theory of Transfinite Numbers (Dover Books on Mathematics), Published by Dover Publications Inc., 2003

7 David Hilbert's Radio Address of 1930
Retrieved 16/12/2021 from YouTube
https://www.youtube.com/watch?v=EbgAu_X2mm4

8 Loeb, D., Sets with a negative number of elements (January 1992)
Retrieved 16/12/2021 from Elsevier
https://doi.org/10.1016/0001-8708(92)90011-9

# How to Play at Quantum Computing (including QKD)

Richard Bornat
School of Science and Technology, Middlesex University, London, UK
richard@bornat.me.uk

Joint work with Rajagopal Nagarajan, R.Nagarajan@mdx.ac.uk

December 2, 2021

**Abstract**

Qtpi is an implementation of a programming language intended to facilitate the exploration by simulation of algorithms for quantum networks. I describe the notation, discuss difficulties of simulating quantum operations, and present a number of examples. There's an appendix describing the basics of matrix quantum calculations.

Quantum mechanics, the theory of quantum physics, is mysterious. Richard Feynman, genius physicist, playfully remarked "I think I can safely say that nobody understands quantum mechanics". This article is not about the mysteries of quantum mechanics but about the simulation of quantum computation,[1] an application of the theory of quantum physics that is relatively recent and will probably have considerable effect. Actual quantum computation is a still-distant dream or an emerging field, depending on how you assess the technical progress that's been made. But Quantum Key Distribution is already a reality: it requires physicists to do little more than send polarised photons to each other and to measure their polarisation, both matters which they are well able to deal with.

Qtpi is an implementation of a programming language, a development of CQP (Gay and Nagarajan, 2005), which allows the description of networks of protocol agents which can perform the quantum operations that are described in theoretical papers – in particular Quantum Key Distribution (QKD) algorithms. It's also able to deal with some simple quantum computation. I have tried to make it Fun to Play With, and I've tried to make this description of it, and what it does, amusing to read. You can get it, ready to run, from github (qtp) for MacOS, Linux and Windows.

I should apologise to the BCS-FACS audience because this paper isn't very formal: mostly it is about how I designed the programming language, although there is a nod to formality in section 2 about qubits as resource. Informality at this point is defensible: qtpi is very much work in progress, and some parts are much more polished than others. I welcome criticism. Indeed this is really more of a sales pitch than a paper: qtpi needs users! Please download it and play with it (or tell me why you won't).

## 1 A language for quantum protocols

Quantum protocols are a major research interest. Quantum Key Distribution is a currently practical network security protocol with two or three agents, Alice cooperating with Bob and perhaps Eve interfering, which

[1] If you are new to quantum computation, you may want to start by looking at appendix A. And then, new readers read on, as the incomprehensible direction in magazines used to have it.

$$
\begin{array}{rcl}
\mathit{procdefs} & ::= & \texttt{proc}\ [\ p(\widetilde{x}) = P\ ]^{+} \\
P & ::= & IO\ .\ P \mid \mathit{qstep}\ .\ P \mid (\mathit{binder})\ .\ P \\
 & & \mid p(\widetilde{E}) \mid (\ P\ ) \mid \mathit{par} \mid \mathit{alt} \mid \mathit{cond} \mid \_0 \\
IO & ::= & C\ \texttt{!}\ \widehat{E} \mid C\ \texttt{?}\ (\widetilde{x}) \\
\mathit{qstep} & ::= & \widehat{E} \gg G \mid q \nearrow (x) \mid q \nearrow [E]\,(x) \\
\mathit{binder} & ::= & \texttt{new}\ \widehat{c} \mid \texttt{newq}\ \widehat{\mathit{qdef}} \mid \texttt{let}\ \mathit{pat} = E \\
\mathit{qdef} & ::= & q \mid q = E \\
\mathit{par} & ::= & [\texttt{|}]\ P\ \texttt{|}\ \ldots\ \texttt{|}\ P \\
\mathit{alt} & ::= & [\texttt{+}]\ IO\ .\ P\ \texttt{+}\ \ldots\ \texttt{+}\ IO\ .\ P \\
\mathit{cond} & ::= & \texttt{if}\ E\ \texttt{then}\ P\ \texttt{else}\ P \mid \texttt{match}\ E\ .\ [\texttt{+}]\ \mathit{pat}.P\ \texttt{+}\ \ldots\ \texttt{+}\ \mathit{pat}.P
\end{array}
$$

$[\,]$ : optional inclusion; $[\,]^{+}$ : one or more;
$\widetilde{s}$ : comma-separated tuple, possibly empty
$\widehat{s}$ : comma-separated tuple, non-empty

Figure 1: The qtpi process language (abbreviated)

### 1.1 Process notation

An abbreviated grammar for the sub-language of qtpi's process notation, as used in the examples before section 9, is given in figure 1: lower case letters for names, upper case for expressions and the like. Qtpi permits a Hindley-Milner typechecker, so although explicit typing is allowed, it is omitted from figure 1 for simplicity.

- `proc` precedes a sequence of mutually-recursive process definitions;
- dot (`.`) is sequencing;
- steps are IO or quantum;
- nothing follows a process activation, so there is only tail recursion;
- nothing can follow a *par*, an *alt* or a *cond*;
- for parsing reasons the null process is represented concretely by `_0`;
- IO steps are sending (`!`) or receiving (`?`);
- a *binder* can bind a name to a new channel, to a new qubit with unknown state, to a new qubit with specified state, or the names in a pattern to classical (non-quantum) values;
- sending requires a channel expression and a tuple of expressions;
- receiving requires a channel expression and a tuple of names, to be bound to the values received (the brackets are not optional);
- gating ($\gg$) puts one or more qubits through a gate;
- measuring ( $\nearrow$ – intended to look like a quantum-circuit meter symbol) takes a single qubit and measures it either in the computational basis or in the basis defined by rotating with a gate;

$$
\begin{array}{rcl}
T & ::= & \texttt{unit} \mid \texttt{bool} \mid \texttt{num} \mid \texttt{bit} \mid \texttt{char} \mid \texttt{string} \mid \texttt{sxnum} \\
 & & \mid \texttt{matrix} \mid \texttt{gate} \mid \texttt{bra} \mid \texttt{ket} \mid \texttt{qubit} \mid \texttt{qstate} \\
 & & \mid \hat{}qT \mid [T] \mid (T, ..., T) \mid T \to T \\
qT & ::= & \texttt{qubit} \mid cT \\
cT & ::= & \text{any type not involving qubit}
\end{array}
$$

Figure 2: Types in qtpi

- a parallel composition is a sequence of processes separated by (|), optionally preceded by (|);
- an *alt* is a sequence of IO guards,[3] each preceding a process, separated by (+), optionally preceded by (+);
- a conditional is an if-then-else, or a pattern match.

The starkness of the pi calculus, which allows formal analysis of process descriptions, is mostly retained in qtpi. The fact that there is no return from a process, that there is no join after the split of a parallel composition, an *alt*, or a conditional, is crucial when building a static check for the correct use of qubits: see section 2.

Qtpi's parser uses offside parsing, so there aren't as many closing brackets in the grammar as you might otherwise suppose.

### 1.2 Types

Figure 2 shows the types of values in the language. Most of them are easily guessed, but

- `num` is unbounded-precision rationals (which of course includes integers);
- `bit` has values `0b0` and `0b1` and is not included in `num`;
- `char` is Unicode characters, and `string` is, as in Miranda, [`char`];
- `sxnum` is symbolic complex numbers, the type of values manipulated by qtpi's symbolic calculator;
- `matrix` is matrices of `sxnum`s;
- `gate` is unitary square matrices, size a power of 2;
- channel types $\hat{}qT$ are restricted so that a channel can carry *either* a qubit *or* a classical value;
- list $[T]$ and tuple $(T, ..., T)$ types.

It looks as if you can make lists of qubits, and indeed you can try, but there's nothing you can do with them afterwards: you can't pass them as function or process arguments, you can't access their elements with functions or with matching. Function types are mostly restricted, but there are exceptions (see below).

[3] When I began work on qtpi, I knew that mixed IO guards in *alt* constructs are very hard to implement in a genuine distributed setting – a problem I fell foul of when developing Pascal-m, even though I found an expensive sort-of-solution (Bornat, 1986). I intended to solve the problem by enhancing channel types with 'write commit' and 'read commit' types but I haven't done that yet. A restriction to allow only read guards, and to prohibit quantum channels in *alt*s, is applied from qtpi version 3.0.

$$\mathit{fundefs} \quad ::= \quad \texttt{fun}\ [\ f\ [\ \widehat{pat}\ ]\ =\ E\ ]^{+}$$

Figure 3: Function definitions

### 1.3 Expression notation

Qtpi has the operators you would expect on numbers and booleans and the like. There are conditional expressions, match expressions and $\lambda$ expressions. The arithmetic operators are heavily overloaded, so that you can, for example, multiply numbers, matrices, gates, numbers with matrices, bras with kets, ... Because juxtaposition $E\ E$ is taken for function application, matrix and gate multiplication has to use an explicit operator ($*$).

Function definitions are included (figure 3): the word `fun` followed by a sequence of mutually-recursive definitions. Qtpi doesn't yet exploit pattern-matching as Miranda does, defining a function with a collection of definitions and taking the first that matches.

There are lots of built-in functions, but apart from two (`qval` and `show`) they can't take non-classical arguments, and none of them delivers a non-classical result.

## 2 Qubits as resource

Thus far all has been relatively straightforward: qtpi is an interpreter for a version of the pi calculus with a library that does matrix arithmetic, and it has a functional notation for doing calculations. Anybody who knows ancient history (Abramsky and Bornat, 1983) would expect me to find it very simple work. Luckily, the difference between qubits and classical bits makes quantum simulation a little bit more interesting.

1. There's a theorem in quantum mechanics (Wooters and Zurek, 1982) that says there's no unitary operation (i.e. no gate or combination of gates) that can clone a qubit with unknown state: you can't start with a qubit in an unknown state $|\phi\rangle$ and one in a known state – $|0\rangle$, say – and finish with two qubits each in state $|\phi\rangle$.

2. Qubits are physical objects, and they are in one place or another: they cannot be shared between processes. (Yes, I know position is somewhat quantum. But in protocols they cannot.)

3. If measurement destroys a qubit (as it might for example destroy a photon) then the measuring agent can't use it again.

4. There is no physical way to discover the actual state of a qubit.

The qtpi language has to be restricted to make sure that violations of these physical restrictions can't be simulated. In principle that's fairly straightforward, but to simplify the explanation, especially the compile-time explanation of errors, I've also imposed some quite strong restrictions on the use of qubits. That allows a static check, rather like a typecheck, for the correct uses of qubits, and requires no run-time checks at all.

### 2.1 No qubits in functions

There's a strict division in qtpi between classical and quantum calculation. I spent a long time down the rabbit hole which leads from qubits in data structures and functions all the way to the gates of Confusion,

before I realised that a swingeing restriction made the description of the language far clearer and a static resource check more straightforward. So functions can't take qubits in their arguments, can't refer to free qubit variables, and can't deliver qubits as results. At a stroke that blocks all kinds of cloning/sharing loopholes and makes it easier to say what qubits an expression depends upon.

### 2.2 No cloning, no sharing

Assignment in an imperative language lets you clone classical values: $x := y$ copies the value from $y$ into $x$. The `let` and `match` constructs in qtpi bind names to values, so they are restricted to classical values just to avoid cloning of qubits.

In the pi calculus channels carry tuples of values, which can include functions and channels. In qtpi we have to be able to send qubits as well. But if a qubit is sent away in a message, using it again in the sending process would mean it had been cloned. Here's a process fragment which attempts that kind of cloning:

```
(newq q) . c!q . Alice(q,c)
```

A qubit is created, sent down channel `c`, and then later provided as argument in a process activation. In syntactic / binding / typecheck terms everything looks fine, provided `c` is in scope and of type `^qubit`. But it's clearly cloning, hoping to share q between the receiving process and Alice.

Qtpi's static check needs to ensure that in a process `c!E,...,E . P`, if the message tuple involves `q`, `P` can't use `q`. But 'involves' and 'use' aren't very precise. To cut a long story short

- A channel can carry *either* a single qubit *or* a classical tuple;
- a message expression can be *either* a single qubit name *or* a tuple of classical values.

This looks severe, but it follows protocol descriptions which talk of 'quantum channels' and 'classical channels' – and, I suspect, that's a distinction forced by physical reality. The restriction makes it simple to discover which qubit is being sent, and it's easy to check whether the succeeding process mentions that qubit or not.

For similar reasons process parameters must be either a qubit or strictly classical. Because we can't reasonably restrict a process to take either a single qubit argument or a tuple of classical arguments, we have to be sure that the same qubit isn't provided twice in the argument tuple, so that the process can't bind the same qubit to more than one parameter name. That means an argument expression must also be a single qubit name or a classical expression.

If a process splits into a parallel composition of sub-processes, it is important to ensure that the subprocesses don't share qubits, because that would be a kind of cloning. CQP handled the problem by using linear types. I didn't understand that treatment at first, and I was keen to use the sort of analysis used in separation logic tools to police the use of shared heap to solve the problem of qubit sharing. So qtpi uses a symbolic execution, a sort of abstract interpretation, to analyse the way that processes use qubits. That's simplified by the simple syntactic structure of the pi calculus and the other restrictions on the language: all you have to do is to look at uses of the names of type `qubit`.

The arms of a *cond* or an *alt* can syntactically share qubit names because only one of them will ever be executed in any particular process elaborationt. But, unfortunately, because it's a static check, the 'mentions' of qubits in a *cond* or an *alt* are the union of the 'mentions' in all of the arms: a necessary blurring.

### 2.3 No life after death

Lots of the protocols that qtpi simulates are, in the real world, to do with polarised-photon qubits. Measuring the polarisation of a photon destroys it in most real-world implementations. Raja was particularly keen that qtpi should recognise this reality. There's a switch '`-measuredestroys`' that can be set true or false: if true, measurement is like sending a qubit away in a message, and you can't use it again. Also, if '`-measuredestroys`' is set true, you can't use a qubit-valued conditional expression in a measurement step, because we need to know which qubit you are destroying.

### 2.4 No comparing

In a functional language you can't compare functions because they simulate infinite relations. In qtpi you can't compare qubits because they simulate physical objects with quantum state and there is no physical way of reading the state of such an object.

In qtpi, this induces a twist. The library function `show`, as in Miranda, converts the value of its argument into a string. To avoid the impossible, it renders functional values as "`<function>`", and to avoid cheating it renders qubits as "`<qubit>`". But sometimes you might want a program to print the state of a qubit – for debugging, say, or just to provide a trace of a calculation.

The solution is the library function `qval`, the type `qstate` and the output channel `outq`. `Qval` takes a qubit and returns a `qstate` value. A `qstate` is a classical value, so you can include it in data structures and pass it to functions and ... but you can't compare `qstate`s. There's actually only one thing you *can* do with a `qstate`, which is to send it down the `outq` channel to be printed. This provides diagnostic printing without leaking secret information, at the expense of some programming clunkiness which you will see in the examples.

Note that there couldn't be a special channel down which you send a qubit to print its state, because once sent down a channel it is lost. Varying the rules for such a channel would be impossible in a static check because channels are values.

## 3 Not infinitely unfair

Choices of partner in a concurrent language raise questions of fairness. Qtpi uses synchronous message-passing, so for example attempting to read from an empty channel causes the reader to wait until some other process offers to write – and vice-versa writing to an empty channel causes the writer to wait until there is a reader. Channels are many-to-many, so a channel can hold many offers before a partner comes along.

Which offer should a partner choose? Strict temporal order of arrival would ensure fairness, but would make a simulation grimly deterministic; one might hope for more random choice, but that raises the possibility of infinite unfairness, when partners might always overlook some offer. In Pascal-m (Abramsky and Bornat, 1983) Steve Cook implemented what he called *lust*: an overlooked offer to communicate has its lust increased, and partners prefer lustier offers. Initial lust values are chosen at random when the offer is made.

Qtpi uses Cook's notion to choose the next process to run at each protocol step, to choose partners in sends and receives, and to choose an arm in an $alt$. Thus it is not temporal-order fair, but neither is it infinitely unfair, and it isn't boringly deterministic. Execution wouldn't be deterministic in many cases anyway, because quantum measurement is probabilistic.

```
proc P(s:^qubit) = + s?(y) . y>>X . s!y . _0
                   + s?(y) . y>>I . s!y . _0

proc Q(x: qubit, s:^qubit) = x>>H . s!x . s?(z) . z>>H . _0

proc System() = (newq q = |0>) (new s:^qubit) | P(s)
                                              | Q(q,s)
```

Figure 4: Coin tossing as a protocol

```
Q #0:|0⟩ >> H; result #0:(h|0⟩+h|1⟩)
0: Q -> P Qbit #0
P #0:(h|0⟩+h|1⟩) >> X; result #0:(h|0⟩+h|1⟩)
0: P -> Q Qbit #0
Q #0:(h|0⟩+h|1⟩) >> H; result #0:|0⟩
```

Figure 5: Sample trace of the cointoss protocol, with P carrying out a flip

use for communication, and splits into two parallel subprocesses, one becoming P, the other Q. P just gets the shared channel, Q gets the qubit and the shared channel.

Q first puts the qubit `x` through the `H` gate (`x>>H`) then sends the modified `x` down channel `s` (`s!x`) and receives a qubit `z` back (`s?(z)`) which it also puts through $H$.

P makes a guarded choice between two subprocesses, one of which receives a qubit `y` (`s?(y)`) and flips it (`y>>X`) then sends the modified qubit back down the same channel (`s!y`); the other receives `y`, doesn't flip it, and sends it back unmodified. Because the guards use the same channel, it chooses with equal probability between the alternatives.[5]

Because P and Q share channel `s` – System gives it to them both as an argument – we can see that Q sends a qubit to P, P receives it from Q, manipulates it and sends it back to Q, and Q receives it again.

This simulation doesn't produce any output, but qtpi can be persuaded to provide a trace of significant events such as exchange of a message and manipulation of qubit states. Figure 5 shows a trace in which P happens to flip the qubit-penny using the X gate. The trace shows that Q puts qubit 0 through an `H` gate ('`#0:|0>`' means 'qubit 0 with state $|0\rangle$'), and shows the result (still qubit 0 with state `(h|0>+h|1>)`, i.e. $|+\rangle$). Then channel 0 carries qubit 0 from Q to P, P puts it through an `X` gate, producing no change, and sends it back to Q. Then Q puts qubit 0 through an `H` gate again, and we finish up with qubit 0 in the same state, $|0\rangle$, that we started with.

If Picard had read Damon Runyon or seen *Guys and Dolls*, he would know about Jacks that jump out of the pack and squirt cider in your ear. The *Enterprise* is doomed.

## 6 Example: teleportation

You can't clone a qubit, and you can't even measure its state, so it was a considerable breakthrough when it was shown (Bennett et al., 1993) that it was possible to reproduce an arbitrary unknown quantum state, but only by destroying the original state at the same time and without learning anything about that original state. The mechanism uses the properties of quantum entanglement (see appendix A). Riefell and Polak (Rieffel and Polak, 2000) describe Alice's part of it accessibly (you don't have to follow the calculations at this point):

[5] This trick, a corrected version of one illustrated in the CQP paper, isn't allowed in qtpi version 3.0 because it uses a quantum channel.

```
proc System () = (newq x=|+>, y=|0>) x,y>>CNot .
                 (new c:^(bit,bit)) | Alice(x,c) | Bob(y,c)

     Alice (x:qbit, c:^(bit,bit)) = (new cz:^qbit)
                                    | Choose(cz)
                                    | cz?(z).
                                      out!["\ninitially Alice's z is "] .
                                      outq!(qval z) . out!["\n"] .
                                      z,x>>CNot . z>>H . z↾(vz) . x↾(vx) .
                                      c!vz,vx .
                                      _0

     Bob(y:qbit, c:^(bit,bit)) = c?(pair) .
                                 y >> match pair . + (0b0,0b0) . I
                                                   + (0b0,0b1) . X
                                                   + (0b1,0b0) . Z
                                                   + (0b1,0b1) . Z*X    .
                                 out!["finally Bob's y is "] .
                                 outq!(qval y) . out!["\n"] .
                                 _0

     Choose (cz:^qbit) =
       (let init = read_bool "initialised qbit or random" "i" "r")
       if init
         then
           (let bv = read_alternative "basis" ","
                        [("0",|0>); ("1",|1>); ("+",|+>); ("-",|->)]
           )
           (newq z = bv) cz!z . _0
         else
           (newq z) cz!z . _0
```

Figure 6: Teleporting an arbitrary quantum state between Alice and Bob

> Alice [...] wants to send the state of [a] qubit $|\phi\rangle = a|0\rangle + b|1\rangle$ to Bob through classical channels. [...] Alice and Bob each possess one qubit of an entangled pair $|\psi_0\rangle = h|00\rangle + h|11\rangle$. The starting state is [a state of 3 qubits]
>
> $$|\phi\rangle \otimes |\psi_0\rangle = \sqrt{\frac{1}{2}}(a|000\rangle + a|011\rangle + b|100\rangle + b|111\rangle)$$
>
> of which Alice controls the first two qubits and Bob controls the last one. Alice now applies $\mathrm{CNot} \otimes \mathrm{I}$ and $\mathrm{H} \otimes \mathrm{I} \otimes \mathrm{I}$ to this state [to give]
>
> $$\frac{1}{2}(|00\rangle (a|0\rangle + b|1\rangle) + |01\rangle (a|1\rangle + b|0\rangle) + |10\rangle (a|0\rangle - b|1\rangle) + |11\rangle (a|1\rangle - b|0\rangle))$$
>
> Alice measures the first two qubits to get one of $|00\rangle$, $|01\rangle$, $|10\rangle$, or $|11\rangle$ with equal probability. Depending on the result of the measurement, the quantum state of Bob's qubit is projected to $a|0\rangle + b|1\rangle$, $a|1\rangle + b|0\rangle$, $a|0\rangle - b|1\rangle$, or $a|1\rangle - b|0\rangle$ respectively. Alice sends the result of her measurement as two classical bits to Bob.

They note that Alice's measurement has destroyed state $|\phi\rangle$, and her qubit is now either $|0\rangle$ or $|1\rangle$. She never knew what $|\phi\rangle$ was, and neither does Bob. But by receiving (0,0) or (0,1) or (1,0) or (1,1) he knows how to switch the amplitudes of his qubit to match what $|\phi\rangle$ was: put it through either I, or X, or Z, or $\mathrm{Z} * \mathrm{X}$ (X and then Z) and he will do the job – but he never needs to learn what $a$ and $b$ are. Magic!

An intriguing part of this description is what Alice does to the qubits. She "applies $\mathrm{CNot} \otimes \mathrm{I}$ and $\mathrm{H} \otimes \mathrm{I} \otimes \mathrm{I}$" to the three qubits, but the protocol only gives her control of two of them. What the description means is that she applies CNot to her two qubits, and the universe (or in our case the qtpi simulator) operates as if the 3-qubit state goes through $\mathrm{CNot} \otimes \mathrm{I}$. Then she applies H to the first of her qubits – the one that used to be $|\phi\rangle$ – and the universe / simulator ensures that the 3-qubit state goes through $\mathrm{H} \otimes \mathrm{I} \otimes \mathrm{I}$.

The simulation, enhanced so that the user can choose what quantum state Alice's qubit must start in, and to include some diagnostic printing, is in figure 6. The System process makes two qubits and entangles them, producing the state $|\psi_0\rangle$. It makes a classical channel for the Alice / Bob communication, then splits into two processes, one which becomes Alice given one of the qubits and the channel, and the other which becomes Bob given the other qubit and the same channel.

Alice starts out by making a new quantum channel on which to receive the $|\phi\rangle$ qubit, and then splits into two processes: one becomes Choose and is given the quantum channel; the other waits to receive a qubit on the quantum channel. The Choose process asks the user what to do, and produces a qubit whose state is either a copy of one of the four standard basis vectors or is genuinely unknown; then it sends that qubit down the channel it was given, on the other end of which is Alice waiting for an answer.

Once Alice has her qubit (in a state which she doesn't know) she first does a bit of diagnostic printing, showing us $|\phi\rangle$ through the `qval`, `outq` trickery. Then she does exactly what Riefell and Polak say she should: put her two qubits through CNot and then just the new one through H, before measuring them both.[6] Finally she sends the measurement results, two classical bits, to Bob, and terminates.

Bob has been waiting patiently on the classical channel for Alice's results, and he puts his qubit through a gate in just the way that Rieffel and Polak described. Then he does some diagnostic printing to tell us what happened so that we can verify, even if he can't, that his qubit is in state $|\phi\rangle$.

Figure 7 shows it teleporting $|-\rangle$; figure 8 shows it teleporting an 'unknown' quantum state. In figure 8 the variables it uses are $a2$ and $b2$ rather than $a$ and $b$, just because the 'unknown' state is created in the qubit numbered 2 in qtpi's simulation. Figure 9 shows the trace of the simulation in the 'unknown' case:

[6] It doesn't matter what order she reads them in, but at present you can't do a multi-qubit read in qtpi. I should fix that.

```
initialised qbit or random (i/r)? i
basis (0,1,+,-)? -

initially Alice's z is #2:(h|0⟩-h|1⟩)
finally Bob's y is #1:(h|0⟩-h|1⟩)
```

Figure 7: Teleporting $|-\rangle$

```
initialised qbit or random (i/r)? r

initially Alice's z is #2:(a2|0⟩+b2|1⟩)
finally Bob's y is #1:(a2|0⟩+b2|1⟩)
```

Figure 8: Teleporting an unknown state

```
1: System (#0:(h|0⟩+h|1⟩),#1:|0⟩) >> CNot; result (#0:[#0;#1](h|00⟩+h|11⟩),#1=#0)

2: Choose -> Alice Qbit #2

3: Alice (#2:(a2|0⟩+b2|1⟩),#0:[#0;#1](h|00⟩+h|11⟩)) >> CNot;
  result (#2:[#0;#1;#2](h*a2|000⟩+h*b2|011⟩+h*b2|101⟩+h*a2|110⟩),#0=#2)

4: Alice #2:[#0;#1;#2](h*a2|000⟩+h*b2|011⟩+h*b2|101⟩+h*a2|110⟩) >> H;
  result #2:[#0;#1;#2](a2/2|000⟩+a2/2|001⟩+b2/2|010⟩-b2/2|011⟩
                       +b2/2|100⟩-b2/2|101⟩+a2/2|110⟩+a2/2|111⟩)

5: Alice: #2: .. as above .. ↾ ;
     result 1 and (#0:[#0;#1](h*a2|00⟩-h*b2|01⟩-h*b2|10⟩+h*a2|11⟩),#1=#0)

   Alice: #0: .. as above .. ↾ ; result 1 and #1:(-b2|0⟩+a2|1⟩)

6: Alice -> Bob (1,1)

7: Bob #1:(-b2|0⟩+a2|1⟩) >> {0  1
                             -1 0 }; result #1:(a2|0⟩+b2|1⟩)
```

Figure 9: Trace of teleporting an unknown state

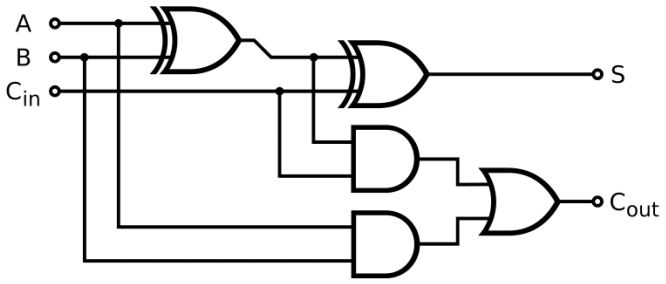



Figure 10: A classical full adder, from (Inductiveload, 2009)

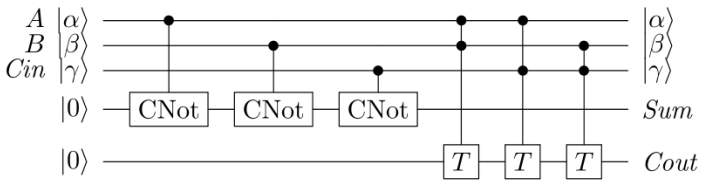



Figure 11: A quantum full adder, adapted from (Coggins, 2020)

1. System entangles qubits 0 and 1 with the CNot gate, giving $h\,|00\rangle + h\,|11\rangle$ ($|\psi_0\rangle$) as required;
2. Choose sends Alice qubit 2 containing the state to be teleported;
3. Alice puts qubits 2 and 0 – her qubit and the first of the entangled pair – through CNot;
4. Alice puts qubit 2 through H;
5. Alice measures qubit 2 and qubit 0, getting (1,1) and putting qubit 1 into state $a2\,|1\rangle - b2\,|0\rangle$;
6. Alice sends Bob (1,1);
7. Bob puts qubit 1 through $\mathrm{Z} * \mathrm{X}$, getting the desired result.

The three-qubit state in figure 9 after step 4 doesn't look like the Rieffel and Polak state because qubit 2 is listed last, whereas in Riefell and Polak the once-was-$|\phi\rangle$ qubit is listed first. Switching first to last, their $|001\rangle$ is $|010\rangle$ in the trace, their $|010\rangle$ is $|100\rangle$, and so on. It is exactly the same state, and after step 5 the other qubit of the entangled pair has exactly the amplitudes described by Rieffel and Polak. Note that it was chance that both of Alice's measurements returned 1: any of the four possible combinations of bits is possible, and the algorithm works in just the way that Rieffel and Polak describe it.

## 7 Example: quantum parallelism

I got quite excited when I realised that you can see quantum parallelism in action using a single-bit full adder because us old-fashioned computer scientists understand full adders. But actually, as Raja pointed out to me, this isn't an example of quantum *supremacy* where a quantum computation gives a result in fewer steps than its classical equivalent.

```
fun basis x =
  read_alternative x "," [("0",|0>); ("1",|1>); ("+",|+>); ("-",|->)]

proc System () =
  (newq qA   = basis "A",
        qB   = basis "B",
        qCin = basis "Cin") .

  out!["full adder sum of (A="] . outq!qval qA . out!["", B="] . outq!qval qB .
  out!["", Cin="] . outq!qval qCin . out![")"] .

  (newq qSum=|0>, qCout=|0>) .
  qA,qSum>>CNot . qB,qSum>>CNot . qCin,qSum>>CNot .
  qA,qB,qCout>>T . qA,qCin,qCout>>T . qB,qCin,qCout>>T .

  out![" is (Cout="] . outq!qval qCout .
  out!["", Sum="] . outq!qval qSum . out![")\n"] .

  qA↗(bA) . qB↗(bB) . qCin↗(bCin) . qSum↗(bSum) . qCout↗(bCout) .

  out!["A measures "; show bA; ", B "; show bB; "; Cin "; show bCin;
       "; Sum "; show bSum; "; Cout "; show bCout; ".\n"] .

  _0
```

Figure 12: Qtpi simulation of figure 11

Figure 10 is a classical logic-gate circuit: two XOR gates compute the sum: $S$ is 1 if one of the inputs $A$, $B$ and $Cin$ (carry in) is 1, or all of them are 1. Two AND gates and one OR gate compute the carry: $Cout$ (carry out) is 1 if there is more than one 1 in the three inputs.

Figure 11 is a quantum full adder,[7] taking $|0\rangle$ as classical 0 and $|1\rangle$ as classical 1, with three CNot and three T (Toffoli) gates. The three CNot gates flip the $Sum$ qubit once, twice or three times according to occurrences of $|1\rangle$ in the three inputs; the result is $|1\rangle$ if there is an odd number of $|1\rangle$s in the inputs. The T gates flip the $Cout$ qubit according to pairs of the inputs, making a $|1\rangle$ out if an odd number of pairs is ($|1\rangle$,$|1\rangle$).[8]

Figure 12 is a program simulating the circuit of figure 11: it reads the user's choice of inputs,[9] creates the $|0\rangle$ values to initialise $Sum$ and $Cout$, and then follows the steps of the circuit diagram. Finally it measures all five qubits. It works as it should, given inputs that are either $|0\rangle$ or $|1\rangle$. For example:

```
A (0,1,+,-)? 1
B (0,1,+,-)? 0
Cin (0,1,+,-)? 1
full adder sum of (A=#0:|1>, B=#1:|0>, Cin=#2:|1>) is (Cout=#4:|1>, Sum=#3:|0>)
A measures 1, B 0; Cin 1; Sum 0; Cout 1.
```

Imitating classical circuits with qubits and quantum gates isn't very impressive, but the circuit of figure 11 has a trick up its sleeve. The qubit state $|+\rangle$, viewed from the computational basis, is the superposition $h\,|0\rangle + h\,|1\rangle$, equally likely to be measured as 0 or 1. An execution of figure 12 in which $B$ is $|+\rangle$ produces

[7] In the online version of this diagram there is a spurious X gate. Perhaps a Mountweazel: at any rate, analysis and experiment confirm that it is unhelpful.

[8] An odd number of pairs is 1 or 3; an even number of pairs is 0 or 2, but if you have 2 pairs then you have 3.

[9] Note that it does this using a function which returns a ket as result. That's ok: a ket is just a vector, not a quantum state.

```
A (0,1,+,-)? 1
B (0,1,+,-)? +
Cin (0,1,+,-)? 1
full adder sum of (A=#0:|1>, B=#1:(h|0>+h|1>), Cin=#2:|1>) is
    (Cout=#4:|1>, Sum=#3:[#1;#3](h|00>+h|11>))
A measures 1, B 0; Cin 1; Sum 0; Cout 1.
```

After initialisation #1($B$) is an isolated qubit, but in the final state, before anything is measured, qubits #1($A$) and #3($Sum$) are entangled: with probability $|h|^2 = 1/2$ they will both measure as $|0\rangle$, and with the same probability they will both measure as $|1\rangle$. In effect the adder has done two additions at once: the case where $B = |0\rangle$ and the case where $B = |1\rangle$. When $B$ is measured it must, with equal probability, deliver 0 or 1 and that will determine the state of the entanglement – measuring $Sum$ afterwards is a formality. The final line shows that in this case $B$ and $Sum$ measured as $|0\rangle$. This outcome occurs, at random, in half of all executions with those inputs; the other half finish, as they should, with

```
A measures 1, B 1; Cin 1; Sum 1; Cout 1.
```

In this execution, as in all others, the final line shows a correct classical addition.

What if all three inputs are in superpositions? We get a five-way entanglement, a superposition of eight possibilities:

```
A (0,1,+,-)? +
B (0,1,+,-)? +
Cin (0,1,+,-)? +
full adder sum of (A=#0:(h|0>+h|1>), B=#1:(h|0>+h|1>), Cin=#2:(h|0>+h|1>)) is
 (Cout=#4:[#0;#1;#2;#3;#4](h/2|00000⟩+h/2|00110⟩+h/2|01010⟩+h/2|01101⟩
              +h/2|10010⟩+h/2|10101⟩+h/2|11001⟩+h/2|11111⟩),
  Sum=#3:[#0;#1;#2;#3;#4](.. the same ..))
A measures 0, B 1; Cin 1; Sum 0; Cout 1.
```

The eight possible classical additions form the entanglement, each with probability $|h/2|^2 = h^2/4 = 1/8$. The program hasn't done any more work than before – six gating steps – but those steps have done eight additions in parallel. Note that there are only eight possible states of the five qubits: the vector has to describe 32 possible states but 24 of them, which don't correspond to correct classical additions, have zero amplitude. The first case of the entanglement, $|00000\rangle$, says that if $A$ measures 0, $B$ 0 and $Cin$ 0 then $Sum$ and $Cout$ must be 0 too, as you'd hope. The measurement in this particular execution found $|01101\rangle$, the fourth case.

But, as I said above, this isn't quantum supremacy. Sure, it's done eight additions in parallel, but measurement only gives you one of them and you have no control over which you get.

## 8 Example: BB84 QKD

Quantum Key Distribution, like quantum teleportation, seems to be a magical trick. Using the quantum properties of a stream of single photons, Alice and Bob generate a random one-time key without disclosing to each other or to anybody else the value of any of its bits. One-time keys are an ideal in cryptography because an attacker can't learn from multiple uses what the key is; one-time keys that don't have to be distributed are the holy grail. (Bennett and Brassard, 1984) describes a QKD protocol, known universally as BB84. It can be simulated in qtpi without difficulty. (Ekert et al., 1992) describes another, based on entanglement and known as E92, which can also be simulated in qtpi. I'll concentrate here on BB84, which is the simpler and the one most often used in today's trial networks.

1. Alice chooses $N$ classical bits at random. She needs many more bits than the size of the message she has to send, as we shall see.

2. Alice sends her bits as $N$ qubits, randomly choosing diagonal basis ($|+\rangle$, $|-\rangle$) or normal basis ($|0\rangle$, $|1\rangle$) for 0 and 1 in each case. She keeps a record of the bases she chose as a second sequence of $N$ bits.

3. Bob receives Alice's qubits one by one and measures each, randomly choosing diagonal or normal bases in each case. He keeps a record of the measurements he made and the bases he chose. If he chose the same basis as Alice he will measure the qubit correctly, provided it hasn't been tampered with en route. If he chooses the other basis, he might not measure it correctly even if there has been no interference.

4. Alice and Bob compare notes about the bases they used (not the values they sent or measured). They use a separate classical channel for this (see below). For the time being suppose that there is no interference on that channel.

5. They each discard the bits for which they chose different bases. If Eve has not interfered, they share $\sim N/2$ secret bits. They haven't disclosed the value of these bits, so they can be used as a one-time code for the message which Alice wants to send.

6. But Eve might have interfered, by intercepting Alice's transmission, receiving, measuring and retransmitting each of the qubits. If she did then (see below) there is a $1/4$ chance in each case that Bob will *not* read the value which Alice sent. To check for interference, Bob chooses a random sample of $n$ of his supposed secret bits – the 'check bits' – and sends them classically to Alice, along with a bitmask to say which bits they are.

7. Alice checks Bob's check bits against her own. There is a $(3/4)^n$ chance that those bits will match even though Eve has interfered. Choose $n$ large enough and Eve is extremely unlikely to get away with it: $n = 100$, for example, gives her only a $3 \times 10^{-13}$ chance of success.

8. If Alice finds that the check bits match, then with high probability she and Bob share $\sim (N/2 - n)$ secret bits. She XORs the message with some of those bits – if she's chosen $N$ large enough there will be enough to do the job – and sends it classically to Bob.

What could go wrong? Let's see. First consider Eve's interference. We can ignore the qubits for which Alice and Bob don't use the same basis, because those will be rejected anyway (step 5). In the case where Alice and Bob guess the same basis, Eve has a 50% chance of making the same guess, reading the qubit correctly and sending an equivalent qubit to Bob, so her interference will be undetected. If she guesses wrongly she will send Bob a qubit encoded in the wrong basis – but Bob might, again with a 50% chance, accidentally measure the 0/1 value that Alice transmitted. So overall she has a $3/4$ chance of not being noticed on each bit. If she can read the classical channel (more on that later) she can even tell which of the bits are going to be discarded and which are used as check bits. If she can evade the check-bit test in step 7) she knows which bits she can rely upon and which not, and with a $1/2$ chance in those bits she might perhaps have a good go at guessing the message. But it's $1 - (3/4)^n$ that she won't evade the test, so Alice almost certainly won't send the message if she interferes: everything depends on step 7.

Note that the effectiveness of the check in step 7 doesn't depend on any argument about the difficulty of computing some function, as classical encryption does. It's rooted in *physical reality*: Eve *cannot* measure a qubit without destroying its state, and that's the basis of the check. It isn't an absolute prevention against Eve's spying, although by choosing large $n$ Alice and Bob can make it statistically very reliable. And in the

```
fun ket_of_bits b v = match (b,v) .
                      + (0b0,0b0) . |0>
                      + (0b0,0b1) . |1>
                      + (0b1,0b0) . |+>
                      + (0b1,0b1) . |->

(* send encoded (bit, value) pairs down channel qc *)
proc SendQbits (bvs,qc,sent) =
    match bvs .
    + []         . sent!() . _0
    + (b,v)::bvs . (newq q = ket_of_bits b v)
                   qc!q    .
                   SendQbits (bvs,qc,sent)

proc Alice ...
  ...
  (new sent)
  | SendQbits(bvs,qc,sent)
  | sent?() .
    (let h0 = hwc bs hks 0 w)
    bsc!h0,bs .                              (* send Bob my bases *)
  ...
```

Figure 13: Sending qubits in BB84 (Alice)

limit it depends on the properties of hardware photon detectors: physical access to a network node can work wonders in that area, and (Chistiakov et al., 2019) reports a successful attack.

But we haven't finished: what about the communication on the classical channel? There are only five classical messages in the protocol, counting check bits and bitmask as separate messages. It's not an ordinary channel: the messages are sent in the clear but individually *authenticated*, each signed with its own one-time hash code, and the authentications are sent along with the message. Eve can read the messages but she can't spoof them, because she doesn't know the hash codes. The authentication scheme (Wegman and Carter, 1981) has very high statistical reliability, depending on the sizes of the hash codes. So Eve can't reliably lie to Alice about what Bob's saying, or vice versa reliably lie to Bob. The authentication hash codes are an initial secret between Alice and Bob, and they can use some of the $\sim (N/2 - n)$ secret bits remaining at step 8 to generate new one-time hash codes for the next round of the protocol. There are even *privacy amplification* mechanisms to generate the codes for the first round from a shared and not-necessarily-entirely-secret password: (Cachin and Maurer, 1997) is only a single reference to a great deal of work in that area.

There isn't much point showing the code of the qtpi simulation, because it's huge and it's almost all classical calculation. I can show you the quantum parts: figure 13 shows how SendQbits, initiated by Alice, generates and sends qubits given a list of (basis,value) bit-pairs: it generates qubits one-by-one from the list `bvs`, and sends them down `qc`, finally sending an empty tuple along channel `sent` to Alice, to say that the job's finished. Her next step is to construct an authenticated message containing her choices of encodings, and send it down channel bsc.

Figure 14 shows that ReceiveQBits, initiated by Bob, generates basis choices and measures the qubits it receives in the corresponding basis, accumulating a reversed list of (basis,value) bit-pairs. Eventually the stream of qubits will dry up and it will receive Alice's list of bases on channel `bsc`, whereupon ReceiveQBits signals the bit-pair list and Alice's message on the `result` channel back to Bob.

Figure 14 is not an entirely believable simulation of Bob's treatment of the photon stream. I wrote it as shown because I wanted only Alice to need to know how big the message is that she has to encode, so that

```
(* receive qbits on channel qc, measure them, return the results
   when you see a tagged message on channel bsc
 *)
proc ReceiveQBits (bvs, qc, bsc, result) =
  + qc?(q)                    .
        (let b = randbit ())
        q↗[if b=0b1 then H else I](v) .
        ReceiveQBits((b,v)::bvs,qc,bsc,result)
  + bsc?(message)             .
        result!(rev bvs,message) .
        _0

proc Bob ...
  ...
  (new result)
  | ReceiveQBits ([], qc, bsc, result)
  | result?(bvs, message1)
  ...
```

Figure 14: Receiving qubits in BB84 (Bob)

```
cd examples/BB84_QKD; time ../../Qtpi functions.qtp Alice.qtp Bob.qtp
   LogAlice.qtp LogBob.qtp SystemAB.qtp
length of message? 1000
length of a hash key? 20
minimum number of checkbits? 40
number of sigmas? 6
number of trials? 1000

... 3721 qbits per trial
all done: 0 interfered with; 1000 exchanges succeeded; 0 failed;
            0 short codebits; average check bits 464.56 minimum check bits 403
... 16.08s user 0.04s system 99% cpu 16.150 total
```

Figure 15: BB84 simulation: Alice communicating directly with Bob

she can calculate $N$ and not bother Bob with it. He only has to notice when she switches from sending qubits to sending a classical bit-list. In practice it's more likely that Alice will send photons at precisely timed intervals, and that the packet size $N$ will be built in to the implementation. In practice Bob might miss some of the photons, for all kinds of physical reasons, but because of the precise timing he will notice when that's happened and can allow for it. I also think it is a bit dodgy to allow an $alt$ which involves a quantum channel. Although it would be quite possible to simulate a more realisable exchange, I think the one depicted here is realistic enough for most purposes. But it's also an illustration of how easy it is to fall into the trap of using classical programming tricks to over-simplify a simulation.

It's important that qtpi can simulate this protocol, with all its paradiddles, correctly and efficiently.[10] Figure 15 shows a summary of the simulation of Alice and Bob without Eve. Alice has to send a message of 1000 bits; to make sure she will have enough secret bits to generate the hash keys and the checkbits and the one-time code, with the chance of running out of secret bits $6\sigma$ away from the mean, she calculates that she needs 3721 qubits per trial. It runs the protocol 1000 times, transmitting about 4M qubits in 20 seconds on a 2021 Mac mini (a cheap small desktop computer, if you are reading this in the far future). The simulation could handle huge messages with ease because it deals with one qubit at a time: this isn't a quantum computation

[10] Full disclosure: for the sake of efficiency, my hash-coding mechanism doesn't properly implement Wegman and Carter's authentication scheme. But then I haven't simulated an attack by Eve on that scheme.

```
(* Read qbits from qA, measure them in a random basis, send the measured bits on qB.
   Stop when you see a message on bsA, and return measurements, tag and message.
 *)
proc CopyQBits (bvs, qA, qB, bsA, results) =
    + qA?(q)            . (let b = randbit ())
                          q↗[if b=0b1 then H else I](v) .
                          (newq q = ket_of_bits b v)
                          qB!q                          .
                          CopyQBits ((b,v)::bvs, qA, qB, bsA, results)
    + bsA?(tag,bits)    . (let bs, vs = unzip (rev bvs))
                          results!bs                    .
                          results!vs                    .
                          results!tag                   .
                          results!bits                  .
                          _0

proc Eve ...
  ...
  (new results)
  | CopyQBits([], qAE, qEB, bAE, results)
  | results?(bs) . results?(vs) . result?(tag) . results?(basesAlice)
  ...
```

Figure 16: Copying qubits in BB84 (Eve)

example, where the space required goes up exponentially with the size of the problem. The simulation checks in the background that Bob actually gets the message that Alice sent, and he does, 1000 times. The simulation uses ∼1/4 of the secret bits as check bits, which is overkill but no matter.

A naive Eve, who just intercepts the qubits and tries to copy them to Bob, but does nothing else, initiates CopyQBits from figure 16. Like ReceiveQBits, it stops when it sees Alice's bases message on the classical channel and sends what it has seen down the `results` channel to a waiting Eve; unlike ReceiveQBits it transmits a copy of each qubit it has read down the channel which Bob reads from. Given the same parameterisation as the Alice-Bob example, a simulation with this Eve between them shows that Alice detects quantum interference on each trial (as she should with 400+ check bits to play with). The simulation takes 26 seconds.

It's possible to play with the parameters to get Alice and Bob to use so few check bits that Eve can sometimes operate unnoticed. It's very artificial because the simulation uses about $N/8$ as the number of check bits, where $N$ is the number of qubits in a trial, and you have to force it to play close to the point where it doesn't generate enough qubits overall to run a trial properly. With a message length of 10, no hash keys, no minimum number of check bits, and no attempt to generate enough bits overall, to achieve 1000 trials it runs out of qubits 718 times, uses an average of only about 3 check bits, and Eve gets away with quantum interference in about half of the completed trials. That's very artificial, but it does show that the security of the protocol is a statistical one.

I'm not a career criminal (it says here) and this isn't the place to discuss possible flaws in this protocol. But I can't resist. One which everybody in the business knows about is the 'trusted node' problem. In a classical network with encrypted communication, intermediate network nodes between sender and receiver are untrusted and just copy the bits they are sent. There's a possibility that a node might be able to decrypt a message, but with modern encodings and large encryption keys it's very remote. In BB84, intermediate nodes have to be trusted to decrypt and re-encrypt the message: each is a Bob connected somehow to a new Alice. That's clearly a point where a criminal might probe. Perhaps another risk is that if an Eve can interpose herself between a trusted Alice and a trusted Bob, mid-network, and if (very big if) she knows

```
fun groverG n = engate ((sx_1+sx_1)*((|+>⊗⊗n)*(<+|⊗⊗n))-(degate I⊗⊗n))

   groverU bs = engate (tabulate_diag_m (2**n) tf
                            where n = length bs
                            where tf i = if i=address then -sx_1 else sx_1
                            where address = bits2num (rev bs) (* big-endian *)
                        )
proc
  System () =
    . (let n = read_min_int 1 "number of bits")
    . (newqs qs = |+>⊗⊗n)

    . (let G = groverG n)
    . (let bs = randbits n)
    . (let U = groverU bs)
    . (let iters = floor (pi*sqrt(2**n)/4+0.5))
    . out!["grover "; show n; " bs = "; show bs; "; ";
           show iters; " iterations"; "\n"]

    . ‖: i←tabulate iters (λ i. i): qs>>>U . qs>>>G . _0

    . qs#(bs')
    . out!["measurement says "; show bs';
           if bs=bs' then " ok" else " ** WRONG **"; "\n"]
    . _0
```

Figure 17: Grover's algorithm in qtpi

the initial set of hash codes for the classical channel, then she can pretend to be a trusted Bob to Alice and a trusted Alice to Bob – i.e. just another node in the network. It's easy to simulate: Eve uses just the CopyQBits code of figure 16 for her quantum self and otherwise just runs the protocol twice. When you simulate it, Eve wins every time, and reads all the messages perfectly, taking 42 seconds on my Mac Mini. Career criminals start here? I dunno.

## 9 And there's more

Real quantum supremacy comes when you can calculate many things in parallel, using the fact that $n$ qubits can simulate $2^n$ classical states (not just could be in one of $2^n$ states, but simulate them all at once), and then you can read out the answer by measuring those $n$ qubits, or sometimes only one of them. The examples I've shown so far work with single named qubits, which is fine for a start but doesn't really cut it when you want to do something more real. Qtpi wasn't intended to deal with quantum computation proper, but with some linguistic innovation it can be pressed into service. There isn't space, you'll be glad to know, to explain the algorithms which I can illustrate.

The most important innovation is *qubit collections*, which are a bit like arrays of qubits. A collection can be indexed to gate an individual qubit, or split into smaller collection, or joined into larger collections, but a collection must be sent and measured as a single unit. That restriction enables qtpi to still run a static resource check but to get the advantages of arrays or lists of qubits, which would otherwise be static-check no-nos. Qubit collections are used to simulate Grover's search algorithm (Grover, 1996)(gro): see figure 17. There's lots of use of overloaded arithmetic operators including exponentiation of kets (`|+>⊗⊗n`) and matrices `I⊗⊗n` There are special operators to gate (`>>>`) and measure (`#`) a qubit collection.

*Iterative processes* are another innovation: Miranda taught me, I believe, that iterative list comprehensions

```
proc W (c,n) =
  if   n<=0 then (let _ = abandon ["W "; show n; " is impossible"]) . _0
  elsf n=1  then (newqs qs = |1>) c!qs . _0
  else . (let k = floor (n/2))
       . (new c1)
       | W (c1,k)
       | . c1?(q0s)
         . out!["W "; show n; " has "; show (n/2); "\n"]
         . (newqs q1s = |0>⊗⊗(n-k))
         . (newq anc = |+>)
         . ‖: i←ixs k: anc,q0s↓i,q1s↓i>>F . out!["."] . _0
         . out!["W "; show n; " has done its Fs\n"]
         . ‖: i←ixs k: q1s↓i,anc>>CNot . out!["."] . _0
         . out!["W "; show n; " has done its CNots\n"]
         . dispose!anc
         . (joinqs q0s, q1s → qs)
         . c!qs
         . _0

proc Wmake (c,n) =
  (let k = powerceiling 2 n)
  | W (c,k)
  | . c?(qs)
    . out!["W "; show k; " = "] . outq!qvals qs . out!["\n"]
    . if k=n then _0
      else
        . out!["discarding "; show (k-n); " qbits "]
        . (splitqs qs → q0s(k-n),qs)
        . q0s#(bs)
        . out!["which measured "; show bs; ", leaving "] . outq!qvals qs
        . if forall (λ b . b=0b0) bs then out!["\n"] . _0
                                     else out![" -- round again!\n"] . Wmake (c,n)


proc System () =
  . (new c)
  . (let n = read_num "how many qbits")
  . Wmake (c,n)
```

Figure 18: Computing a W state in qtpi

were better at expressing operations on lists than recursion can be. Qtpi's iterative processes are heavily restricted, of course, to support the static check on qubit and qubit collection resource. There's an iterative sub-process in figure 17 – it's the line starting '. ‖:' in the System process which takes the list $0..iters-1$ and for each element, gates the qbit collection qs through U and then G. It does so in two steps because gating through G*U wouldn't be able to exploit the diagonal nature of U – oh the joys of matrix hacking!

The foundation of the algorithm is the two gates G and U, built by the groverG and groverU functions: U is diagonal and mostly 0, but G is mostly $h^{2n-2}$. This is a proper quantum computation example, and the time and space it takes grow exponentially with the size of `qs`. Nevertheless, qtpi on my Mac Mini can handle 18 qubits, which require moderately large matrices and 402 iterations of the quantum steps, in 75 seconds. Time is the limitation in this example, rather than space: simulating 19 qubits requires 569 iterations and takes 270 seconds (about four times longer) but uses only 338 MB of memory (about four times more).

I spent a good part of the first Covid-19 lockdown of 2020 improving qtpi's treatment of sparse matrices, including functional representation of some matrices. That helped a lot with Grover's algorithm, but it

helped even more with an algorithm that creates W states, an entanglement in which in every possible state only one qubit is $|1\rangle$ and the rest are $|0\rangle$ (Dür et al., 2000). Figure 18 is adapted from an algorithm in Microsoft's Quantum Katas (Microsoft, 2020). There are two iterative sub-processes in the W process, each of which indexes a collection (`q0s↓i` and `q1s↓i`). Once it's produced an entanglement, W joins its two collections together with `joinqs`.

If you ask for a number of qubits that isn't a power of 2, the algorithm has to make too many and throw some away, using a `splitqs` binder and a collective measurement of the discarded bits (`q0s⫽(bs)`) in the Wmake process. If the measurement shows `q0s` as all $|0\rangle$, then the $|1\rangle$ qubit must be in `qs`. If not, the simulation has to try again, which happens a little less than half the time in the worst case.

Qtpi can create a W state of 1024 qubits, on my 64-bit-address-space Mac mini, in 30 seconds. That means that it has to use matrices which are up to $2^{1024} \times 2^{1024}$: well outside the computer's address range. Functional sparse matrices rule!

As to Shor's algorithm, which factors integers and which people always ask about: I don't understand it yet. As soon as I do, I'll have a go at simulating it.

## 10 Conclusion

Qtpi is a simulator and calculator for quantum protocols and smallish quantum computing problems. You can get it, with examples and documentation, from `github.com` (qtp). Please get it, play with it, and please tell me whether or not it Brings You Simple Joy Of Calculation.

## References

Copenhagen interpretation. URL `https://en.wikipedia.org/wiki/Copenhagen_interpretation`. 24

Grover's algorithm. URL `https://en.wikipedia.org/wiki/Grover's_algorithm`. 20

Qtpi quantum simulator. URL `https://github.com/mdxtoc/qtpi/releases`. 1, 22

Samson Abramsky and Richard Bornat. Pascal-m: a language for loosely-coupled distributed systems. In Y. Paker and J.P. Verjus, editors, *Distributed Computing Systems (Synchronisation, Control and Communication)*, pages 163–189. Academic Press, 1983. 5, 7

John S Bell. On the Einstein Podolsky Rosen paradox. *Physics Physique Fizika*, 1(3):195, 1964. 26

C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers, Systems, and Signal Processing*, page 175, India, 1984. 15

C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W K Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters*, 70 (13), 1993. 9

Richard Bornat. A protocol for generalized occam. *Software: Practice & Experience*, 16(9):783–799, 1986. 4

Christian Cachin and Ueli M Maurer. Linking information reconciliation and privacy amplification. *journal of Cryptology*, 10(2):97–110, 1997. 17

Vladimir Chistiakov, Anqi Huang, Vladimir Egorov, and Vadim Makarov. Controlling single-photon detector id210 with bright light. *Optics express*, 27(22):32253–32262, 2019. 17

Macauley Coggins. How to perform addition on quantum computers, 2020. URL `https://thequantumdaily.com/2020/01/10/quantum-programming-101-how-to-perform-addition-on-quantum-computers/`. 13

Wolfgang Dür, Guifre Vidal, and J Ignacio Cirac. Three qubits can be entangled in two inequivalent ways. *Physical Review A*, 62(6):062314, 2000. 22

Albert Einstein, Boris Podolsky, and Nathan Rosen. Can quantum-mechanical description of physical reality be considered complete? *Physical review*, 47(10):777, 1935. 26

Artur K. Ekert, John G. Rarity, Paul R. Tapster, and G. Massimo Palma. Practical quantum cryptography based on two-photon interferometry. *Phys. Rev. Lett.*, 69:1293–1295, Aug 1992. doi: 10.1103/PhysRevLett.69.1293. 8, 15

S. J. Gay and R. Nagarajan. Communicating quantum processes. In *32nd Symposium on Principles of Programming Languages (POPL 2005)*, pages 145–157, 2005. doi: 10.1145/1040305.1040318. Also arXiv:quant-ph/0409052. 1, 2

Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996. 20

Inductiveload. Circuit diagram of a full adder, 2009. URL `https://upload.wikimedia.org/wikipedia/commons/archive/6/69/20140323122652%21Full-adder_logic_diagram.svg`. 13

N David Mermin. Could Feynman have said this? *Physics Today*, 57(5):10, 2004. 24

David A Meyer. Quantum strategies. *Physical Review Letters*, 82(5):1052, 1999. 8

Microsoft. Q# kata on superposition, task 16, WState_PowerOfTwo_Reference, Aug 2020. URL `https://github.com/microsoft/QuantumKatas/tree/main/Superposition/ReferenceImplementation.qs`. (visited 2020/10/08). 22

Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i. *Inf. Comput.*, 100(1): 1–40, 1992. 2

Eleanor G. Rieffel and Wolfgang Polak. An Introduction to Quantum Computing for Non-Physicists. *ACM Comput. Surv.*, 32(3):300–335, 2000. Also arXiv:quant-ph/9809016. 9, 24

Mark N Wegman and J Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of computer and system sciences*, 22(3):265–279, 1981. 17

W. K. Wooters and W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299(802):16–23, 1982. 5

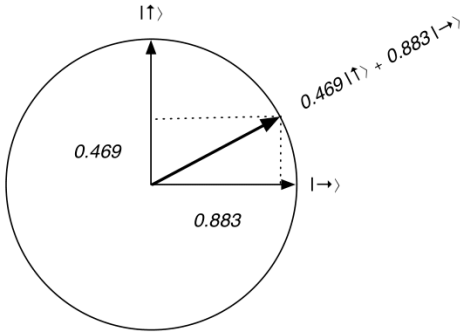



Figure 19: Quantum state: a vector in a unit circle

# A Shut Up and Calculate

Quantum mechanics is an extremely effective description of very much of what we know about the universe. Physicists and philosophers debate what it means: in particular the observable phenomenon of entanglement and the matter of quantum measurement remain knotty scientific and philosophical problems. The Copenhagen interpretation, developed by Heisenberg and Bohr in the early twentieth century and admirably summarised on Wikipedia (cop) has been wittily summarised by Mermin (Mermin, 2004) as 'shut up and calculate'. Qtpi is a calculator. I shall try, in this discussion, to shut up about What It All Means, and write **DDM**, for Deep Dark Mystery, where I don't want to say too much in case I get it wrong.

A good place to start is where I started, with Reifell and Polak's "Introduction to Quantum Computation for Non-Physicists"(Rieffel and Polak, 2000). Everything in this appendix can be found there, and more, but I think it worthwhile to give enough detail here to underpin the examples in the paper.

Quantum mechanics is vector and matrix arithmetic. It's that simple, though of course why it works as a description of the real world – and it *does* – is a DDM.

## A.1 The basics: qubits, quantum states, measurement

A **qubit** is a physical object with a quantum property, such as a photon with plane or circular polarisation or an atom with spin, or even position if it's quantised. It doesn't matter what quantum property you choose: they all obey the same algebra.

The **quantum state** of a single isolated qubit can be seen as a vector: see, for example, the heavy arrow in figure 19. By convention the vector is length 1, and it can then be described by two **amplitudes**, projections onto orthogonal **basis vectors**, here shown as $|\uparrow\rangle$ and $|\rightarrow\rangle$. In the figure the vector is at $62°$ to the vertical, and the amplitudes are given as approximations to $\sin 62°$ and $\cos 62°$. In Proper Quantum Mechanics (and also in qtpi since Raja insisted) the amplitudes are complex numbers, but for the purposes of the examples in this paper you can stick with real numbers. Complex amplitudes are a DDM.

Every state can be written as the sum of basis vectors, each multiplied by an amplitude, that is as a **superposition** of the basis states. For a single isolated qubit superpositions are that simple, but it gets a little stranger with entanglement.

If you pick different basis vectors then the same quantum state – the same vector – will have a different description: see, for example, figure 20. Just rotate your basis vectors – your view of the state – and the amplitudes change. Note that this makes superpositions very unmysterious: a basis vector in figure 19 is a superposition in figure 20, and vice versa.

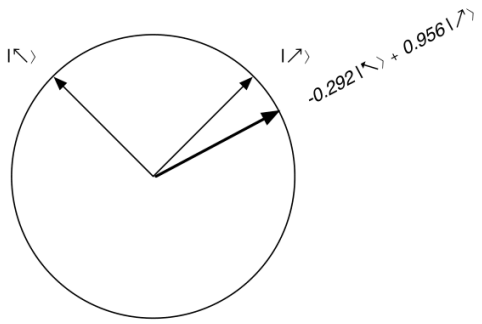



Figure 20: Quantum state: same state as figure 19, different basis vectors

**Quantum measurement** is mysterious, and there's no getting round the fact. To measure the state of a physical qubit you must use a physical device, and the device must use two orthogonal basis vectors. The measurement will, with probabilities which are the squares of the amplitudes of the vector in that basis, report one basis vector or the other – a binary result - and it will **change the quantum state** to that basis vector. Ouch! There is **absolutely no way** to measure the state of a qubit if it isn't a basis vector already. Ouch! again.

The squares of the amplitudes, if they were real numbers as in my example, would by Pythagoras's theorem sum to the length of the vector, which is 1. Because they are complex numbers in general, we use the norm squared: $|z|^2 = zz^*$, where $z*$ is the complex conjugate[11], so $|x + iy|^2 = x^2 + y^2$. The probability of measuring $|\uparrow\rangle$ if the quantum state is $a\,|\uparrow\rangle + b\,|\rightarrow\rangle$ is $|a|^2$, and of measuring $|\rightarrow\rangle$ is $|b|^2$.

Actually the length of the vector in a quantum state doesn't matter: multiplying by a scalar, even a negative scalar, makes no difference to the ratio of $|a|^2$ and $|b|^2$. That means that $+|1\rangle$ and $-|1\rangle$ are the same quantum state, and that means that even when a state is a tensor product, not an entanglement, it can't be uniquely decomposed.

Dirac's bra-ket notation is universally used, and has already been abused in the paragraphs so far. A **bra** is a row vector and is shown as $\langle ...|$; a **ket** is a column vector and is shown as $|...\rangle$. Quantum states are kets. The bra $\langle x|$ is the conjugate transpose[12] of $|x\rangle$ and vice-versa. You can do standard vector arithmetic with bras and kets – see, for example, figure 17.

The **computational basis** is the pair $\{|0\rangle,|1\rangle\}$, where $|0\rangle$ is the column vector $\binom{1}{0}$ and $|1\rangle$ is $\binom{0}{1}$. The **diagonal basis** is the pair $\{|+\rangle,|-\rangle\}$ where $|+\rangle$ is the column vector $\begin{pmatrix}\sqrt{\frac{1}{2}}\\ \sqrt{\frac{1}{2}}\end{pmatrix}$ and $|-\rangle$ is $\begin{pmatrix}\sqrt{\frac{1}{2}}\\ -\sqrt{\frac{1}{2}}\end{pmatrix}$. Qtpi's symbolic calculator, like this paper, represents $\sqrt{\frac{1}{2}}$ with the symbol $h$ (nothing to do with Planck).

The quantum state of **multiple qubits** is a $2^n$-component column vector. Its basis vectors are the tensor product[13] of the individual states' basis vectors: a 3-qubit state in the computational basis, for example, has basis vectors $|000\rangle$, $|001\rangle$, $|010\rangle$, ..., $|110\rangle$, $|111\rangle$. If the qubits aren't entangled (we'll come to that), the

[11] Complex conjugate: flip the sign of the imaginary part, so $(x + iy)^* = x - iy$.

[12] Conjugate transpose: transpose and flip the signs of the imaginary parts.

[13] The *tensor product* of a vector $x_{0..n-1}$ and a vector $y_{0..m-1}$ is like $n$ copies of $y$, each multiplied by a value from $x$.

$$(x \otimes y)_{im+j} = x_i y_j$$

state overall is the tensor product of the individual states.

### A.2 Entanglement

Hold on, said Einstein, Podolsky and Rosen (Einstein et al., 1935): this doesn't make physical sense. They had spotted what they thought was a bug: it seems possible that a pair of qubits, with basis vectors $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$, could be in the state $h\,|00\rangle + h\,|11\rangle$ – 50% chance of both being $|0\rangle$, 50% chance of both being $|1\rangle$, 0% chance of anything else. It would seem that if you measured the first qubit and found $|0\rangle$, and then measured the second, it would have to be $|0\rangle$ too; or if the first measured $|1\rangle$ then so would the second. The time between the measurements, and the distance between the sites of measurement, doesn't seem to matter. 'Spooky action at a distance', Einstein called it. Can't be right.

But it is right. Bell (Bell, 1964) showed that EPR states are one way to distinguish quantum physics from what had come before, and experiments have proved that reality seems to work the quantum way, with the experimental time between measurements so short, and the distance between sites so wide, that there can't be any communication even at the speed of light. Spooky indeed, and why it works is still a DDM.

It's actually really easy to produce the EPR state in quantum computing, using CNot and H gates, and those states play a significant rôle in quantum algorithms. They are called **entanglements** because you can't take them apart: there are no two-element kets $|x\rangle$ and $|y\rangle$, for example, such that $|x\rangle \otimes |y\rangle = h\,|00\rangle + h\,|11\rangle$.

### A.3 Gates

**Quantum gates** are **unitary** matrices of complex numbers: square matrices $M$ such that $MM^*$ (where $M^*$ is the conjugate transpose of $M$) is a unit matrix, with 1s on the diagonal and 0s everywhere else. (This property makes quantum gates **reversible**, which is a DDM.) Quantum gates for a single qubit are $2 \times 2$; quantum gates for $n$ qubits are $2^n \times 2^n$. That means that the memory and time needed to simulate computation with $n$ qubits grow exponentially with $n$. Oh dear!

What gates do is to rearrange the amplitudes in a quantum state: they can be applied to a state which is a basis vector, of course, but they do most of their work on superpositions. Nevertheless, it can be helpful to read a quantum gate in terms of what it does to basis vectors.

Here are some single-qubit gates:

$$\begin{array}{lll} \mathrm{I}: & \begin{array}{l}|0\rangle \to |0\rangle \\ |1\rangle \to |1\rangle\end{array} & \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} \\ \mathrm{X}: & \begin{array}{l}|0\rangle \to |1\rangle \\ |1\rangle \to |0\rangle\end{array} & \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix} \\ \mathrm{Z}: & \begin{array}{l}|0\rangle \to |0\rangle \\ |1\rangle \to -|1\rangle\end{array} & \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix} \\ \mathrm{H}: & \begin{array}{l}|0\rangle \to |+\rangle \\ |1\rangle \to |-\rangle\end{array} & \begin{pmatrix}h & h\\ h & -h\end{pmatrix} \end{array}$$

I (identity) leaves the amplitude multiplying $|0\rangle$ alone, and likewise the amplitude multiplying $|1\rangle$ – i.e. it makes no change. X (exchange) swaps the amplitudes. Z negates the amplitude multiplying $|1\rangle$, which doesn't do anything about the probabilities the vector encodes but may affect future calculations. In the H (Hadamard) matrix I write $h$ for $\sqrt{\frac{1}{2}}$: it rotates the state from the computational to the diagonal basis. Of all of these H is the most interesting: it's its own conjugate transpose, so (remember that $\mathrm{H}\,\mathrm{H}^* = \mathrm{I}$) it also rotates from the diagonal to the computational basis. And $\mathrm{H}^{\otimes\otimes n}$, the tensor product of $n$ copies of H, is

such that $(\mathrm{H}^{\otimes\otimes n})_{i,j} = (-1)^{i\&j}h^n$, which means that it's easy to represent as a functional sparse matrix ($\mathrm{I}^{\otimes\otimes n}$ is even easier, of course).

Applying a gate to one of the qubits in a multi-qubit state, even an entanglement, is possible: the chosen one goes through the gate and the rest go through I. So to apply a single-qubit gate $G$ to the $n$th qubit in an $m$-qubit state you apply $(\mathrm{I}^{\otimes\otimes(n-1)}) \otimes G \otimes (\mathrm{I}^{\otimes\otimes(m-n-1)})$.

Two- and three-qubit (and ...) gates exist. Two that are used in the examples in the paper and are held to be useful are CNot (2 qubit) and Toffoli (3 qubit):

$$\text{CNot}: \quad \begin{array}{l} |00\rangle \to |00\rangle \\ |01\rangle \to |01\rangle \\ |10\rangle \to |11\rangle \\ |11\rangle \to |10\rangle \end{array} \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\text{Toffoli}: \quad \begin{array}{l} |000\rangle \to |000\rangle \\ |001\rangle \to |001\rangle \\ |010\rangle \to |010\rangle \\ |011\rangle \to |011\rangle \\ |100\rangle \to |100\rangle \\ |101\rangle \to |101\rangle \\ |110\rangle \to |111\rangle \\ |111\rangle \to |110\rangle \end{array} \quad \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

CNot swaps the $|10\rangle$ and $|11\rangle$ amplitudes. Sometimes this is described as flipping the second bit $|0\rangle \longleftrightarrow |1\rangle$ if the first is $|1\rangle$, but actually it is more powerful than that, because it swaps the amplitudes of the $|10\rangle$ and $|11\rangle$ states: that aids parallelism in the full adder (section 7). Toffoli does a similar thing, but with $|110\rangle$ and $|111\rangle$.

# LMS-FACS Evening Seminar 2021

## *Underpinning Mainstream Engineering with Mathematical Semantics*

Professor Peter Sewell
University of Cambridge

November, 2021

Reported by: Rob Heirons

## Synopsis

Despite 80+ years of research on semantics and verification, mainstream computer systems and their engineering development processes remain almost entirely non-formal, reliant on ad hoc testing and prose specifications. These have been good enough for industry to thrive, but their inability to exclude errors is one of the root causes of today's endemic security failures.

In this talk, I discuss what it takes to put mathematically rigorous semantics to work for full-scale mainstream systems, touching on scientific, engineering, and social aspects, and on the benefits and costs. This draws on work with many colleagues on various key interfaces: processor architectures, programming languages, and network protocols; and on the CHERI and Morello projects, extending conventional architectures and languages with hardware support for capabilities, for fine-grained memory protection and encapsulation.

Taking mainstream engineering artefacts seriously also prompts new theory and tools, e.g., for the relaxed shared-memory concurrency semantics of real machines, quite different from traditional concurrency semantics, and for the semantics of C and of CHERI capabilities.

## About the speaker

Peter Sewell is a Professor of Computer Science at the University of Cambridge. His research aims to enable rigorous semantics-based engineering of mainstream systems, including real-world concurrency semantics, instruction-set semantics, and CHERI. His PhD was with Robin Milner in Edinburgh. He has held ERC AdG, EPSRC, and Royal Society research fellowships. With Watson, Moore, and Arm, he was one of the instigators of the UKRI Digital Security by Design programme, supporting development of the Arm Morello prototype CHERI Armv8-A architecture, processor, software, and semantics.

On Thursday 19th November 2021, Professor Peter Sewell, from the University of Cambridge, gave the annual FACS/LMS talk. The event was hybrid, with approximately ten people in attendance physically and over 80 online. I believe that this was also the first hybrid talk to be held at De Morgan House. The format worked well, allowing many questions from both the physical and online audiences.

The title of the talk was "Underpinning mainstream engineering with mathematical semantics". Within this, Professor Sewell described research in areas such as network protocols, relaxed memory concurrency in hardware (processors) and relaxed memory concurrency in programming languages. Within these, he discussed a number of common problems. In most cases, there was no formal specification and the natural language descriptions available were imprecise and of relatively little value. In addition, these systems are extremely complex and there is a need for any formalism to cope with this complexity.

The work described addressed these problems by devising a mathematical language (formalism) that was specifically designed for the problem domain and that could also be mechanised through, for example, a theorem prover. Specifications were built through what was essentially an experimental process: by producing a specification based on observations. The complexity of the systems considered led to observations being entirely black-box and so specifications described interfaces. An important benefit of this approach was that the (mechanised) specifications could act as test oracles.

Professor Sewell finished the talk with a description of some of the work being carried out in the CHERI project. This project, which involves academics and researchers from Arm, aims to design a processor that provides guarantees regarding (the absence of) certain types of vulnerabilities associated with memory usage. The work is thus different from the previously described research since semantics were used at the design stage. Professor Sewell largely concentrated on one of the features of CHERI, which is to enrich pointers with additional information about how they can be used, allowing these requirements to be checked in real-time.

The talk was recorded:

https://www.youtube.com/watch?v=LeqbCTdsXOo

and the original announcement with details is given here:

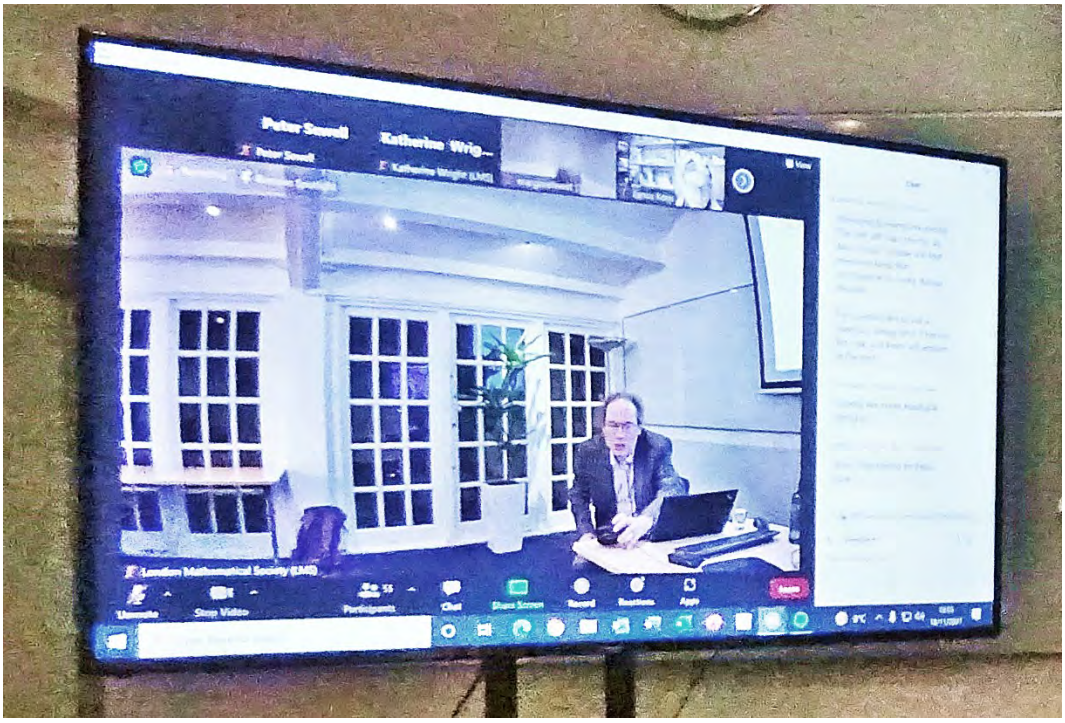
https://www.bcs.org/events-calendar/2021/november/lmsfacs-talk-underpinning-mainstream-engineering-mathematical-semantics/

*Rob Hierons introducing Peter Sewell's talk.*

Many people contributed to the success of the event. I would like to thank Professor Sewell for accepting our invitation and for given an informative and thought-provoking talk. The support of both the BCS-FACS committee and the LMS Computer Science committee, led by Professor Bowen and Professor Wong respectively, was also crucial. Finally, I would like to thank Katherine Wright for her patience and for organising the event.


*Peter Sewell at the start of his talk.*


*Peter Sewell, explaining the importance of solid foundations in hardware and software!*



*Remote access via Zoom, including comments and questions by online participants.*

All photographs by Jonathan Bowen

# *SEFM 2021 Conference Report*

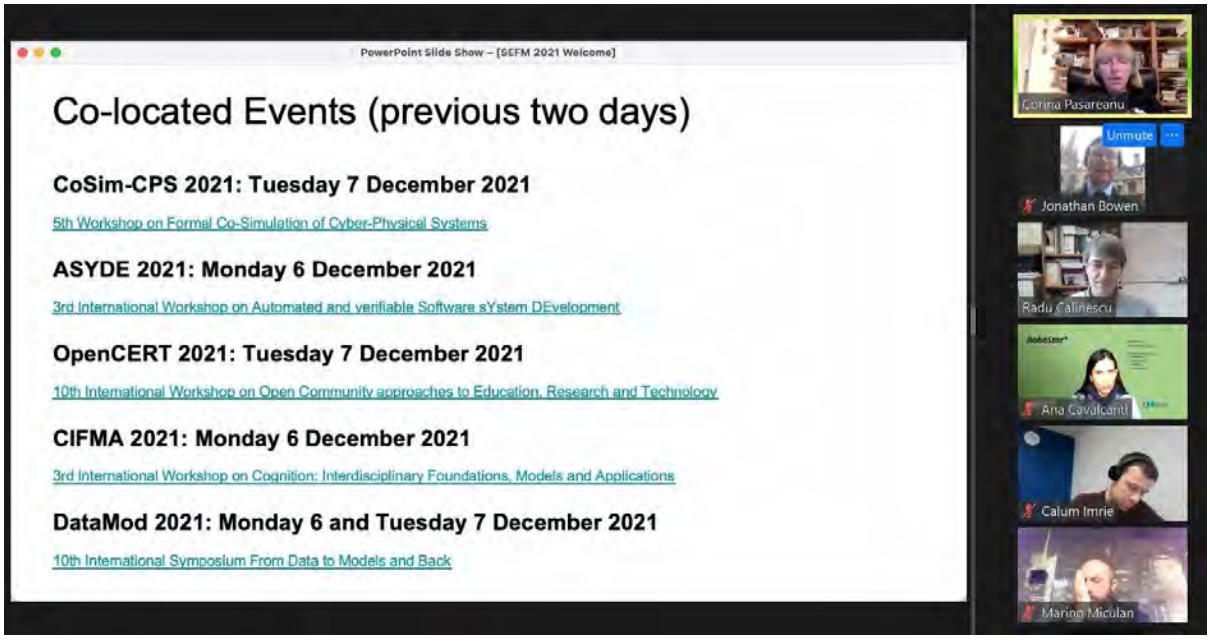
Jonathan Bowen
Chair, BCS-FACS,

December 2021

## Overview

The SEFM 2021 *19th International Conference on Software Engineering and Formal Methods* (https://sefm-conference.github.io) was held entirely online due to the Covid pandemic during 6-10 December 2021, with free registration. The conference was jointly organised by Carnegie Mellon University (USA), Nazarbayev University (Kazakhstan), and the University of York (UK).

During 6-7 December, a number of online one-day workshops and a two-day symposium were held, including OpenCERT 2021, *10th International Workshop on Open Community approaches to Education, Research and Technology* (https://opencert.github.io), at which I presented a paper on *Formal Methods Communities of Practice: A Survey of Personal Experience* (Bowen, 2021b). This was partly inspired by Egon Börger's 75th *Festschrift* earlier in the year in association with the ABZ 2021 conference (Bowen, 2021a), which I attended with a paper presentation online. See also Tim Denvir's review of the *Festschrift* proceedings elsewhere in this issue of *FACS FACTS*. A number of FACS members attended the OpenCERT talk online. A Springer LNCS post-proceedings is planned.

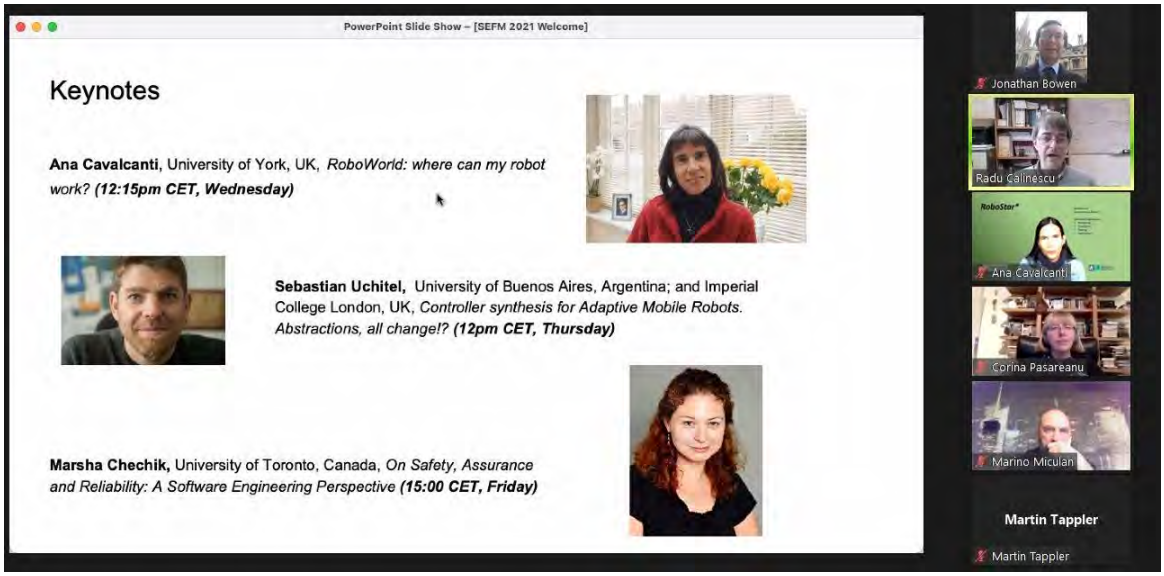



*Co-located workshops and symposium with the SEFM 2021 conference.*

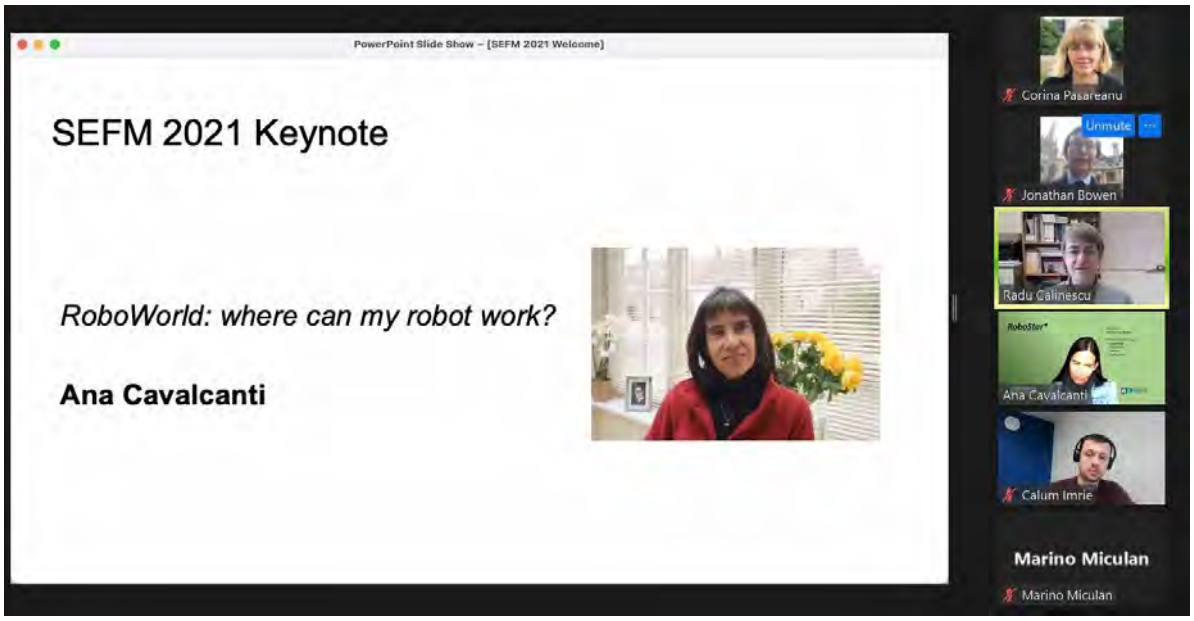
The main SEFM 2021 conference was held during 8-10 December, with a keynote speaker on each day of the conference. I attended a number of the talks, including a keynote talk by Ana Cavalcanti, University of York, chair of Formal Methods Europe (FME) the FACS FME Liaison Officer, on *RoboWorld: where can my robot work?*

The SEFM 2021 proceedings has been published in the Springer LNCS series (volume 13085), appearing in time for the conference (Calinescu and Păsăreanu, 2021).


*SEFM 2021 opening page on Zoom.*




*Keynote speakers at the SEFM 2021 conference.*

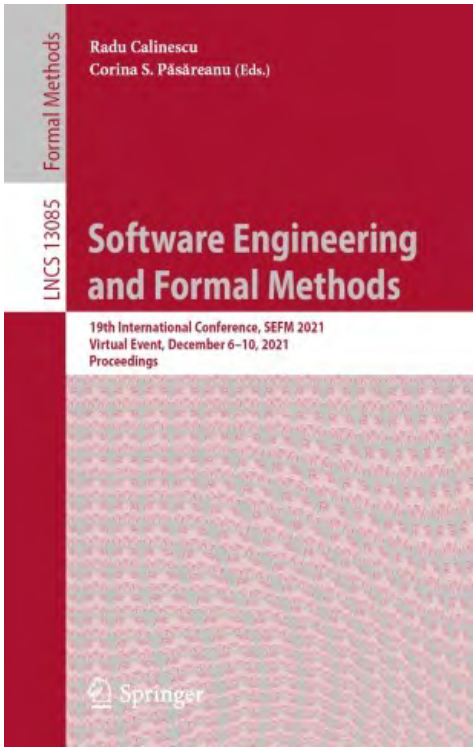



*Ana Cavalcanti's keynote talk at SEFM 2021.*

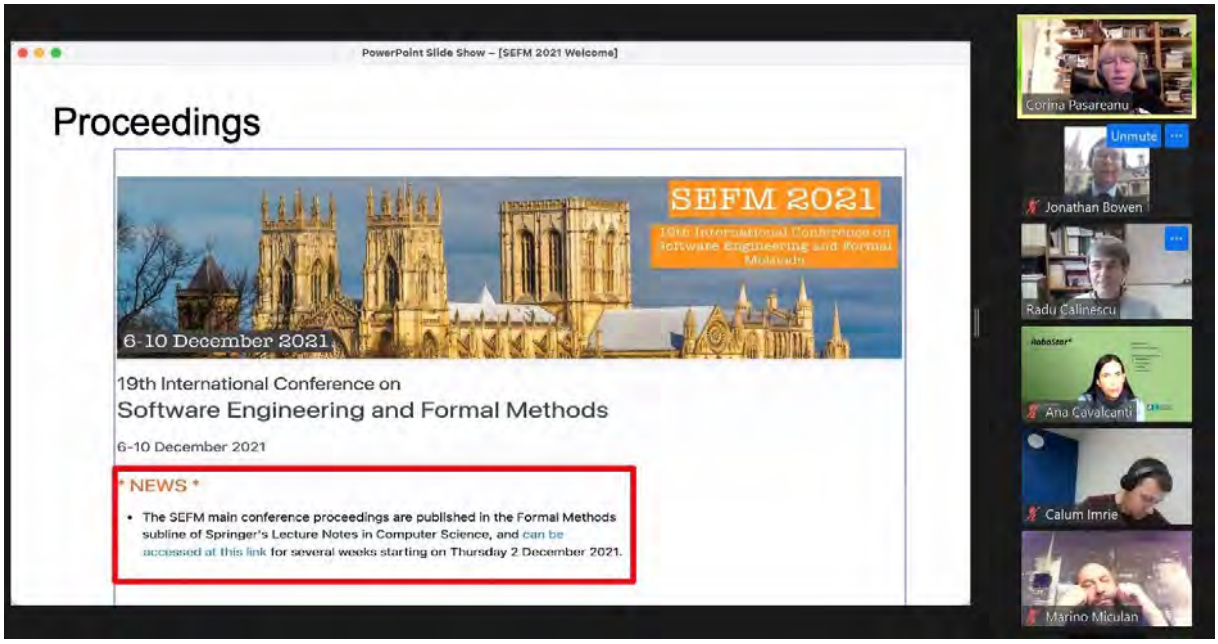

*The opening slide for Ana Cavalcanti's keynote talk.*

# Conclusion

The SEFM 2021 conference and its associated workshops went well in the circumstances, being forced to be completely online in the current pandemic circumstances. An advantage is that it is easy for anyone to attend from all over the world, with no travel costs and free registration. The ease of recording online talks also means that that they can be made available for later listening. A significant disadvantage is that personal networking is much less effective online, meaning that future international research collaborations are more difficult to foster. From personal experience, starting new research cooperation is much easier with someone that one has already met in reality. A physical conference is an ideal opportunity for this. After this initial contact, collaboration online is easier. In addition, for online conferences, there are timetabling issues with speakers and the audience in different time zones around the world.




For further information on SEFM 2021, see: https://sefm-conference.github.io



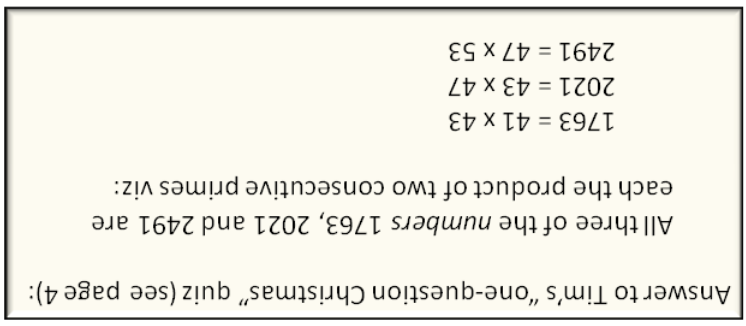

*Information on the SEFM 2021 proceedings*

## References

Bowen, J.P. (2021a). ABZ 2021 Conference Report. *FACS FACTS*, 2021-2, pp. 65—70, July 2021. URL: https://www.bcs.org/media/7577/facs-jul21.pdf

Bowen, J.P. (2021b). Formal Methods Communities of Practice: A Survey of Personal Experience. In *OpenCERT 2021*, 7 December 2021. URL: https://opencert.github.io/2021/Pre-proceedings/Bowen-Breuer.pdf

Calinescu, R. and Păsăreanu, C. S. (eds.) (2021). *19th International Conference, SEFM 2021, Virtual Event, December 6–10, 2021, Proceedings*. Springer, Lecture Notes in Computer Science, volume 13085. DOI: 10.1007/978-3-030-92124-8



# Short Reports: *CALCO 2021, MFPS 2021, and FMAS 2021*

August—September, 2021

Reported by: Margaret West

John Cooke and I registered for a hybrid co-located event viz:

> *Conference on Algebra and Coalgebra in Computer Science (CALCO 2021)*

and:

> *37th Conference on the Mathematical Foundations of Programming Semantics (MFPS 2021)*

which took place as a hybrid event from Salzburg, Austria between August 30th and September 3rd 2021. See https://www.coalg.org/calco-mfps2021/

There was a very interesting programme of papers from both Conferences which also included a City walking Tour of Salzburg for those who attended in person. The walk was transmitted via a head camera to those of us who attended online.

---

> *Third Workshop on Formal Methods for Autonomous Systems (FMAS 2021)*

This two day workshop proved to be both interesting and stimulating took place between 21st and 22nd October 2021 and was hosted by Maynooth University, Ireland. This can be accessed via its web site: https://fmasworkshop.github.io/FMAS2021/

The workshop presented recent work on formal verification of autonomous systems and was intended in particular to stimulate collaboration between the robotics and formal methods communities. There were two invited speakers by, respectively, Clare Dixon and Divya Gopinath.

This web site includes a list of accepted papers which will subsequently be available online via http://www.eptcs.org.

The invited talks are now available for everyone on the FME youtube channel:
https://www.youtube.com/channel/UC5rZj0AyBudca0YRgEAX-Ow/videos

**Clare Dixon:** https://www.youtube.com/watch?v=qRzROixTcEY&t=8s
**Divya Gopinath:** https://www.youtube.com/watch?v=mIJ3yVVy_BM&t=3s

# Annual BCS-FACS Landin Memorial Seminar 2021

## *Making Concurrency Functional*

Glynn Winskel
Huawei Edinburgh

December 2021

Reported by: Brian Monahan

One of the most challenging issues within computing lies in understanding systems that involve concurrency. Many regard the problem as concerning how to provide some kind of *concurrent programming model* that can enable developers to reliably design systems involving a wide range of interacting concurrent agents and their behaviour. Such an approach typically needs to provide not only an appropriate notation but also, crucially, a well-defined semantic foundation. That foundation must show how the notation defines agents and must also describe how they may interact and evolve their configurations as a result.

There are many different approaches to specifying concurrent behaviour, typically starting from a core notation which is then endowed with a behavioural semantics describing computational reduction, interaction and communication, typically using something like a *Structural Operational Semantics* involving inference-style rules and bisimulation equivalences, as in the case of notations like CCS and π-calculus, or by using a bespoke domain of traces and failures, such as for CSP.

Glynn Winskel's starting point is very firmly the other way about – his work starts out, not with some notation for concurrent behaviour, but instead begins by investigating fundamental mathematical structures which naturally lend themselves to an exploration of interaction and dependency and therefore what it means for system behaviour to be concurrent. As he states, his approach provides "a maths-driven foundation based on distributed/concurrent games based on event structures, with interaction by composition of strategies".

At the start of the seminar, Winskel related the story of the sole one-to-one meeting he had with Peter Landin himself. Winskel had approached Landin to discuss studying for a PhD with him. During this rather informal meeting, the topic of concurrency came up, with Landin proclaiming that he would start by considering concurrency in terms of "dependencies between the scopes of variables". Although Winskel went instead to Edinburgh to study the semantics of concurrency with Gordon Plotkin, it turned out that Landin's earlier remark concerning dependency proved to be rather prescient. Winskel's thesis work developed the notion of *Event Structure,* based upon dependency and conflict relations between events, which in turn provided the mathematical basis for a semantic characterisation of Petri Net behaviour in domain theoretical terms.

The remainder of the seminar presented a high-level tour of Winskel's mathematical approach to concurrency semantics. Broadly speaking, the basic ideas explored here consider games to correspond to types/constraints and strategies to correspond to programs/processes. Winskel started by reviewing the computational aspects of functions with respect to interaction and in particular focused upon input and output dependencies. The issue here concerns how one might desire functions to take further inputs and also provide additional outputs in rather a piecemeal fashion, so as to represent interaction and communication between concurrent agents in a natural, compositional way. Clearly, this cannot work as such, but nevertheless contains a valuable, core idea – how might it be possible to describe concurrent behaviour in a way that is fundamentally *compositional* so that concurrent systems can be readily built from independent parts using compositional combinators that naturally encapsulate concurrent actions and behaviour. If such an approach could be found, it would provide a *naturally compositional* approach to concurrent systems and their design (i.e. in the same sense that pure functions are naturally compositional for functional programming and design). The subtle difference between this approach and the process algebraic approach is that process algebra *imposes* structure on a pre-defined syntax structure *via* use of operational semantics rules and reductions, whereas with this approach, the semantic relationships between entities are not directly imposed but instead emerge naturally as inherent properties of entities having a well-formed mathematical definition.

Event structures were then formally introduced as providing the basic structure upon which Winskel's approach to concurrency semantics is constructed – this introduces a couple of fundamental relations over events that represent *causal dependency* and *conflict*, satisfying a couple of simple properties.

The next big step is to introduce a structure which conveys what is intuitively meant by *interaction* and this is the formal notion of (2-person) *distributed game*, based upon separate work, initially by John Conway [1] and then taken in a more categorial direction by André Joyal [2]. The insights behind this notion of game are fundamentally combinatorial and structural in nature where games involve making discrete moves to change some underlying configuration.

More notions and structures briefly touched upon in the rest of the seminar include the important notion of *Event Structures with Polarity,* together with the notion of *strategy*. *Event Structures with Polarity* importantly provide a way to represent 2-party games between Process (as Player) and Environment (as Opponent) through alternating markings. A *strategy* (for Player) in a game is then a choice of moves for Player together with their *causal dependence* on Opponent moves – and symmetrically for strategies for an Opponent. The important operation of taking the "dual" of a game is then defined (by swapping Player and Opponent), together with the notion of parallel composition which purely amounts to simple juxtaposition.

Further definitions and notions were also introduced, including ways to enhance and extend behavioural characteristics to include quantum, probabilistic and real-number

behaviour. The level of mathematical sophistication displayed here necessarily increases with the introduction of various significant constructions taken from Category Theory. These higher-level constructions help provide compositional tools that also operate at a lower functional level.

The overall message of this excellent seminar is that this approach yields a purely mathematical framework of distributed games and strategies that can be specialised to functional approaches that can be enhanced to incorporate a wide-range of applied computational phenomena, such as quantum, probabilistic, and real-number aspects, and thus includes applications such as back-propagation for machine learning.

Finally, the meeting itself was held over Zoom with a good number in attendance (well over 40 people). A video of the Zoom meeting will be made available in due course via the BCS – together with the slides for this talk.

For anyone wishing to follow up further with Winskel's research work and his approach then there is also an introductory paper by Winskel in the previous FACS newsletter [3] which goes to some length in describing many of the concepts briefly touched upon in this talk. Many of Glynn Winskel's research papers can also be found online at https://www.cl.cam.ac.uk/~gw104/.

## References

[1] J.H.Conway, On Numbers and Games, 2nd Ed, A K Peters/CRC Press; (11 Dec. 2000)

[2] A.Joyal.: *Remarques sur la theorie des jeux a deux personnes*. Gazette des sciences mathematiques du Quebec, 1(4) (1997). English translation 2003 by Robin Houston.

[3] G.Winskel, Domain Theory and Interaction, FACS FACTS newsletter, Issue 2021-2, July 2021

## Postscript

In an earlier draft shared with Glynn Winskel, I had a sentence saying that there was *no* connection between work on distributed games and economic game theory – but that turns out to be incorrect. There has been some research work to explore the connections between the two areas – here are some relevant references:

Clairambault, P., Gutierrez, J., Winskel, G.: **The winning ways of concurrent games**. In: LICS 2012: 235-244 (2012)

Winskel, G.: **Winning, losing and drawing in concurrent games with perfect or imperfect information**. In: Festschrift for Dexter Kozen. Volume 7230 of LNCS., Springer (2012)

Hedges, J: **Dialectica Categories and Games with Bidding**, In Post-proceedings of TYPES'14. Leibniz International Proceedings in Informatics 39:89-110, 2015

Hedges, J, **A first look at Open Games**, Blog post, https://julesh.com/2017/09/29/a-first-look-at-open-games/, 2017

Neil Ghani, Jules Hedges, Viktor Winschel, Philip Zahn: **Compositional Game Theory** In LICS 2018: 472-481 (2018)

Glynn Winskel further observes that:

> *There's a lot of overlap [between economic and concurrent game theory] and I believe a great deal of traditional game theory can be subsumed under concurrent games. A start is made in the paper with Clairambault on Concurrent games with payoff—e.g. we show optimal strategies compose; Hedges's open games can be presented as parameterised dialectica games and through concurrency, multiple players can be expressed etc.*
>
> *More work is needed of course …*


*First slide of Glynn Winskel's seminar.*


*Glynn Winskel answering questions at the end of his seminar.*

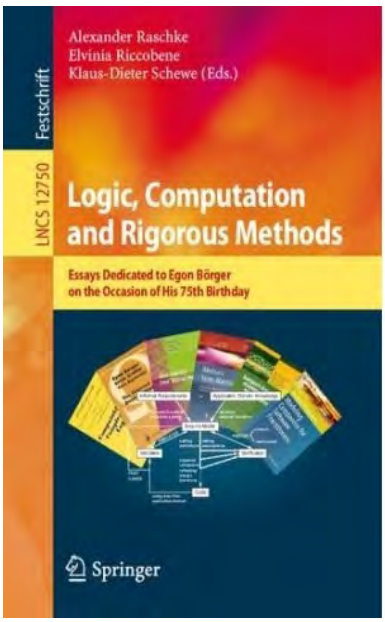
(Screenshot by Jonathan Bowen)

# Book Review: *Essays Dedicated to Egon Börger on the Occasion of His 75th Birthday*

***(Editors:** Alexander Raschke, Elvinia Riccobene, Klaus-Dieter Schewe)*

Reported by: Tim Denvir


*Egon Börger.*
*(Frontispiece of the Festschrift, from the ABZ 2016 conference website)*




This volume, number 12,750 in the LNCS series, starts with an account of Börger's life and work by the editors. They give a wide-ranging summary of Börger's academic history, posts, and significant publications.

Börger began his academic life in "philosophy" but this seems to have consisted mainly in logic, indeed mathematical logic. They assert "he spent two decades in logic and three decades in computer science".

The editors' introduction is clearly written with great care and noble attempts at rhetoric. For example: "Börger is known not only as an excellent scientist but also as a virtuoso for playing with the dialectic antipodes of theory and practice". They describe how Börger originally wanted to go into music and become a conductor, but was persuaded to study philosophy instead. This he did at two of the most prestigious French universities, the Sorbonne and Louvain.

An early monograph dwelt on using formal language to express semantics (meaning), fact and problems. The editors are highly critical of “official politics” at the Universität Münster which, they say, prevented Egon Börger succeeding Dieter Rödding after the latter’s unexpected death and, they claim, resulted in the university’s loss of reputation in computer science. One assumes that Dieter Rödding was head of the computer science department. Börger spent four out of five of his sabbaticals with industrial companies. The authors emphasise Börger’s research and involvement with Abstract State Machines and the many publications and organisations in that area which he spearheaded.

There are 18 papers in the volume, by 44 authors. Many papers relate to Abstract State Machines (something Börger himself was, as already noted, heavily involved with) but many other topics are covered: Domain Knowledge (there are references to Michael Jackson, Pamela Zave, Dines Bjørner); Mitotic sets in computability theory; Cyber-physical systems; Stepwise Design; Non-monotonic reasoning; Optimisation; The Requirements Process; B and Event-B; Model Theory; Safety Assurance; Business Process Management. Throughout, the papers relate their subject matter to semantics and almost all to ASMs. Several of the papers initially take the form of letters to Egon Börger. In all, this volume is an exuberant wide-ranging sincere tribute to Egon Börger and would be a pleasure, I think, to possess.

## Picture Credit

Cover photo and photo of Egon Börger with kind permission from the organisers of the Festschrift celebrated by LNCS 12750 and ABZ 2016.

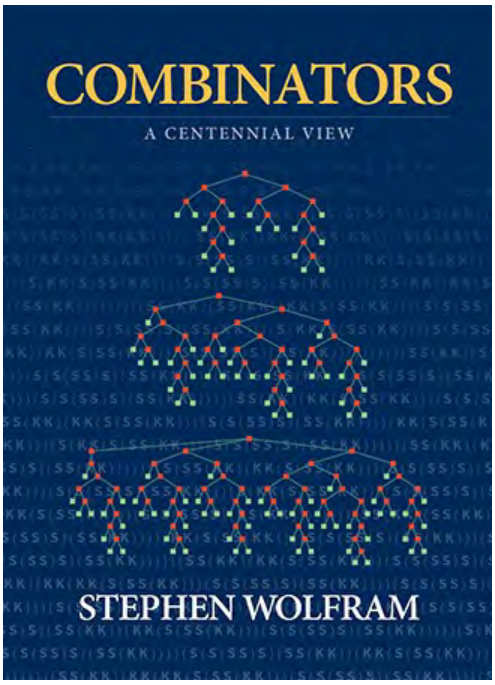
## Reference

LNCS 12750 Logic, Computation and Rigorous Methods: Essays Dedicated to Egon Börger on the Occasion of His 75th Birthday. Springer Nature Switzerland 2021. DOI: *10.1007/978-3-030-76020-5*

# Book review: *Combinators: A Centennial View*

By Stephen Wolfram
Wolfram Research

Reported by: Jonathan P. Bowen




A previous article in *FACS FACTS* (Bowen, 2021) has discussed Russian logician Moses Schönfinkel (1888 – 1942) and his pioneering work on combinatory logic, prompted by an online talk by Stephen Wolfram (2020a), exactly a century to the hour after a talk by Schönfinkel in Germany, where he was a member of David Hilbert's group. Wolfram wrote further on this in several December 2020 blog posts on his *Writings* website (Wolfram 2020b). These have now been expanded into the book under review (Wolfram 2021).

Wolfram is in the enviable position of not having to worry about research proposals from funding bodies or research assessments. The success of his eponymous company and its products, especially the *Wolfram Mathematica* mathematical software tool, means that he can pursue research interests in whatever direction he wishes. Although a mathematician, he has other collaborators with complementary research skills that can help with archival research to put his mathematical interests into a historical context with newly discovered historic documents. He is essentially a modern-day "gentleman scientist" (now known more gender-neutrally as an "independent scientist"), just as others like Aristotle, Charles Babbage, Charles Darwin, and Albert Einstein were in the past, with enough personal resources, and of course expertise and skill, to pursue his scientific interests as he wishes.

The book starts with three major sections, based on Wolfram's online writings (Wolfram, 2020b), followed by three shorter sections, an annotated bibliography, and a rather small print but comprehensive index. The first large 160-page section has the same name as the book and sets the overall scene regarding mathematical aspects of combinators. The entire book is 362 pages long in total (321 pages without the bibliography and index), so this section alone is around half of the book. The examples use the "Wolfram Language", a text-based functional language that enables mathematical modelling and is the basis of *Mathematica*. The material is extensively illustrated with visualizations. The section notes that combinators pre-date the concept of a Turing machine, lambda calculus (both conceived in 1936), and even Gödel's incompleteness theorems (published in 1931). Schönfinkel's ideas on combinators presented in 1920 are probably the first examples of abstract universal computation,

as often modelled by the notion of a universal Turing machine. If the ideas had been more widely publicized at the time, theoretical computer science could have taken a different course. Combinators as introduced in 1920 are built from two basic combinators, now known as **s** and **k**, which can be modelled as simple replacement rules, namely, using the notation of the Wolfram Language:

**s[x_][y_][z_] ⟶ x[z][y[z]]**

**k[x_][y_] ⟶ x**

Wolfram presents many examples using patterns in his Wolfram Language to build a variety of complex objects. As a specific case, he demonstrates how the 16 different 2-input Boolean functions (or "gates" if you are a computer hardware engineer) can be modelled with just combinations of applying **s** and **k**. For example, the *Nand* function/gate ("not and", which can be used in combination to create any Boolean function) can be modelled as the rather more complicated looking:

**s [s [k [s [s [s] [k[k[k]]] ] ] ] ] [s]**

with **k[k[k]]** representing *True* and **k[k[s[k]]]** representing *False*. Although the **s**/**k** expressions can look complicated to the human eye, this does not detract from the universality with which they can be used for modelling computation.

The section continues with even more complex examples using the Wolfram Language, with various visualizations of these. These include application to chemical-like "molecules". The complexity of some of the patterns generated is reminiscent of Turing's later work on morphogenesis, based on deceptively simple mathematics to produce perhaps unexpected shapes and patterns (Bowen et al., 2018).

The second 46-page section is entitled *Combinators and the Story of Computation*. This takes a more historical view of early developments with respect to combinators, using archival documents, many of which are newly discovered and presented in their historical context. A significant number of these relate to Moses Schönfinkel himself. Others are related to Haskell Curry (1890–1982), who studied for his PhD at the University of Göttingen, the same university where Schönfinkel was based and presented his combinator ideas in 1920. Curry adopted and developed Schönfinkel's approach. He had the opportunity to publish more widely than Schönfinkel. Hence, we now have the notion of "Currying" in computer science (converting a multiple-argument function into a sequence of functions with a single argument), whereas it could have become known as "Schönfinkeling" (Bowen 2020), although the latter would perhaps trip off the tongue less easily!

The third 65-page section entitled *Where Did Combinators Come From? Hunting the Story of Moses Schönfinkel* and an associated 14-page addendum entitled *Where Did Combinators Come From? Hunting the Story of Moses Schönfinkel* provide more detailed information on Schönfinkel's life and work, with interesting historical evidence through extensively researched documents sought from a range of sources by Wolfram's team.

The main section ends with the connections to Curry's related research. The addendum provides more information on Schönfinkel's later life, which has been somewhat shrouded in mystery. It includes information on Moses's younger brother, Gregory Schönfinkel, who lived in Moscow. This may be the reason that Moses ended up there. There is little to go on with respect to Moses's mental condition and even his precise date of death (which could be 1940 or 1942). There are claims that Moses lived in a "communal apartment", but this may have been Gregory's apartment. Overall, the addendum makes some progress, and it is possible to speculate, but there is still much doubt about Moses's later life and death. The addendum ends with some thoughts on the work of Kazimierz Ajdukiewicz, who overlapped with Moses Schönfinkel in David Hilbert's group, and later produced some research work with similarities to combinations, but with no explicit mention of Schönfinkel. It is assumed that they must have interacted, but there is no documentary evidence for this.

A short section announces a $20,000 prize for an "S Combinator Challenge". Wolfram hypotheses the conjecture that the rule

$$S\ f\ g\ x \longrightarrow f[x][g[x]]$$

is sufficient to achieve universal computation by applying it repeatedly. Wolfram believes that this may well be the case but has not proved it. If anyone reading this review can prove or disprove this conjecture, then the BCS-FACS Specialist Group would be delighted to invite that person to give a joint seminar with the London Mathematical Society or in the annual Peter Landin Semantics Seminar series. Please contact the author of this review in this case!

A further section includes relevant excerpts from Wolfram's 2002 book *A New Kind of Science*. A comprehensive bibliography related to combinators in various subsections is provided, together with a helpful index, including names of the wide range of people mentioned in the book.

Overall, the first section of this book is relevant to those interested in theoretical computer science and later sections are relevant to those interested in the history of computing and mathematics. Although there is no formal dedication page at the start of the book apart from the appreciation in the preface, this volume provides an apt memorial for Moses Schönfinkel, who deserves to be better known as a foundational figure in computer science. Had his work and life not been cut short by sad circumstances of later mental health issues and lack of support, this might have already been the case. Unlike Turing, his status has not yet been elevated to an appropriate level compared to his foundational achievement regarding the introduction of the idea of combinators. This book goes some way to address that in both a mathematical and historical context.

# References

Bowen, J. P. (2021). Moses Schönfinkel and combinatory logic. *FACS FACTS*, 2021-1:23 – 25, February. URL:https://www.bcs.org/media/6694/facs-feb21.pdf

Bowen, J.P., Trickett, T., Green, J. B. A., and Lomas, A. (2018). Turing's Genius – Defining an apt microcosm. In J. P. Bowen, J. Weinel, G. Diprose, and N. Lambert (eds.), *EVA London 2018 Proceedings*, pp155 – 162. BCS, Electronic Workshops in Computing (eWiC). ScienceOpen. DOI: 10.14236/ewic/EVA2018.31

Wolfram, S. (2002). *A New Kind of Science*. Wolfram Media. URL:https://www.wolframscience.com/nks/

Wolfram, S. (2020a). *Combinators: A 100-Year Celebration*. YouTube, 7 December 2020. URL:https://www.youtube.com/watch?v=PG2G5xSz0NQ

Wolfram, S. (2020b). Posts from 2020: December 2020. *Writings*. URL:https://writings.stephenwolfram.com/2020/12/

Wolfram, S. (2021). *Combinators: A Centennial View*. Wolfram Media. ISBN: 978-1-57955-043-1 (hardback), 978-1-57955-044-8 (eBook). URL:https://www.wolfram-media.com/products/combinators-a-centennial-view.html

**History of Computing Collection at Swansea University**

The History of Computing Collection specialises in computing before computers, formal methods, and local histories of computing. An introduction to the Collection appeared in the February 2021 issue of *FACS FACTS* (2021-1, pp.10-17). The Collection is located on the Singleton Campus of Swansea University; it can be visited by appointment. A small number of items from the Collection are on display in the Computational Foundry, Bay Campus, which is the home of the Computer Science Department. All inquiries welcome.

***From the History of Computing Collection, Swansea University:***

# *Unfinished Business: Abstract Data Types and Computer Arithmetic*

John Tucker
University of Swansea

The emergence of abstract data types is something of a landmark in the theory and practice of programming. The basic idea is simple and beautiful: An abstract data type is characterised by its operations and tests together with a list of their algebraic properties, thought of as axioms or laws. Implementation details are hidden from the abstract data type's user. It is an abstraction that supports design, comprehension and reasoning.

In the formal theory, the operations and tests are specified syntactically in a signature $\Sigma$ and their properties are specified in a set $E$ of equations, or equations with conditions, that the operations and tests are required to satisfy. Implementations are modelled by certain algebras of signature $\Sigma$ satisfying the laws in $E$. Two implementations are equivalent if they are isomorphic as $\Sigma$ algebras.

The abstract data type is a landmark notion of the 1970s for in it we see many programming notions, such as: *interfaces, axiomatic specifications*, *information hiding*, *correctness*, *generics*, etc. – combined and made precise, in a way that is conceptually strong enough to support an enduring and surprising mathematical theory. The History of Computing Collection has plenty of material on the birth of abstract data types. Here I am going to look at their ancestry.

I have chosen two offprints from the Collection that are themselves landmarks in the study of data types. Although they belong to 'prehistory', thinking about their legacies reminds us of unfinished business for computer science, and serious unfinished business at that.

## Von Neumann and Goldstine on computer arithmetic

The first is an offprint of a truly fundamental paper by John von Neumann (1903-1957) and Hermann Goldstine (1913-2004),

> Numerical inverting of matrices of high order
> *Bulletin American Mathematical Society*, 53: (11), (1947), 1021-1099.

It belonged to the Leslie J Comrie (1893—1950), a doyen of numerical computation before computers, and was part of the library of his company Scientific Computing Services Ltd, which now belongs to the Swansea Collection (Figure 1). Comrie founded the company in 1937 following his dismissal from the Admiralty. In the company's prospectus he explained:

> I am endeavouring to offer the scientific public an entirely new service – namely computations of a scientific or technical nature, done by trained professional computers using, whenever possible, the calculating and accounting machines that are now available.

The company is regarded as the first *scientific* computing bureau.

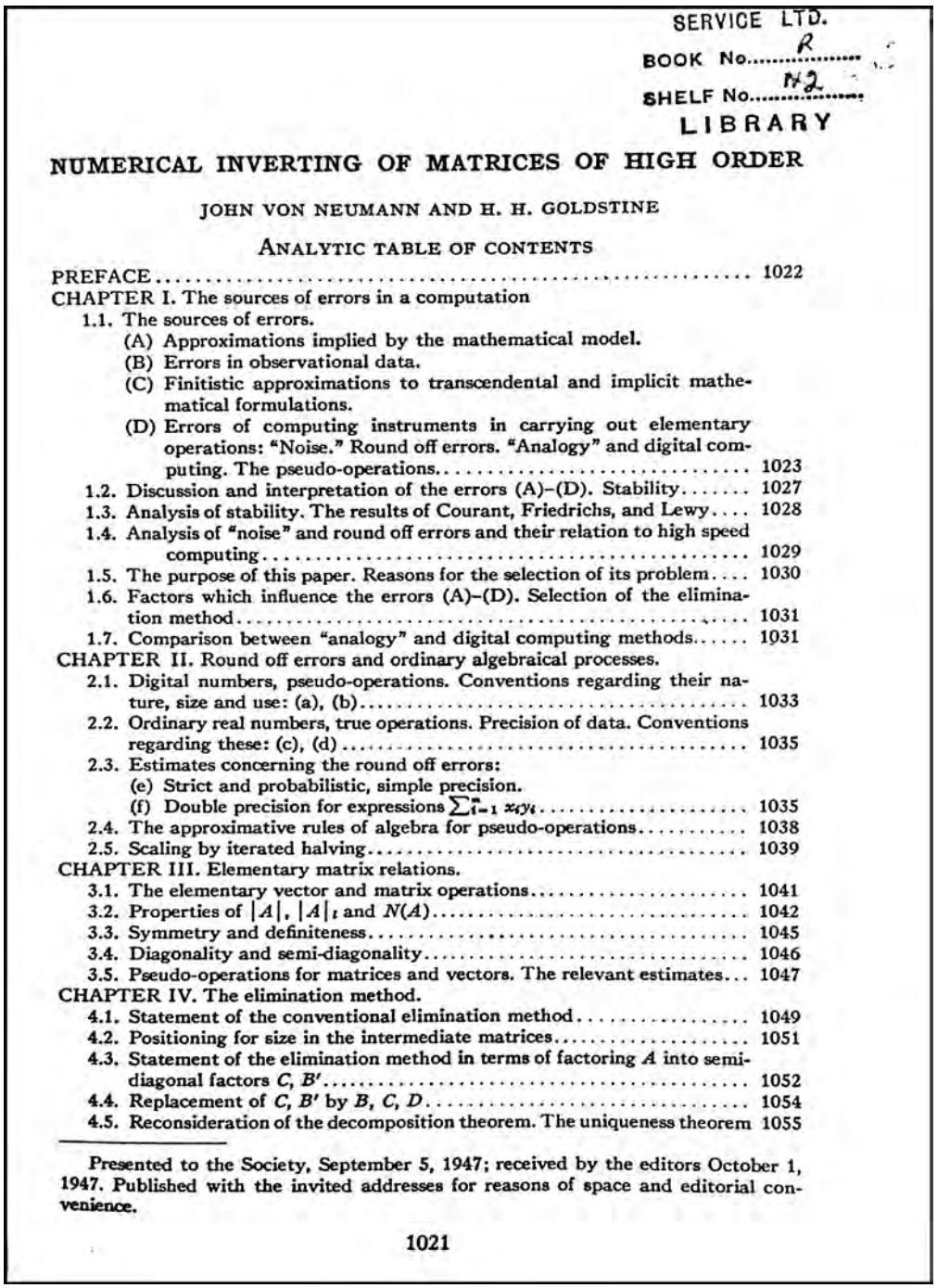
Ostensibly, the von Neumann and Goldstein paper are about calculating the inverse of a (positive definite) matrix on an electronic computer, a task necessary for the solution of large sets of linear simultaneous equations. The importance of solving linear systems, and the classical method of Gaussian elimination, cannot be overestimated, then or now. But this contribution of November 1947 is much more.

Solving linear systems, including very large systems, was long known and had been the subject of serious technical worries about rounding errors, not only by engineers but by contemporary statisticians and economists such as the Harold Hoteling (1895 - 1973), whose analysis of the Gaussian Elimination method of 1943 give a bound of $4^{n-1}$. In our paper, von Neumann and Goldstine removed the pessimism surrounding solving large sets of linear equations, and set a standard for the error analysis of algorithms that stood for decades. However, the standard was so high that it cast a shadow over the development of the mathematical analysis of rounding errors!

Von Neumann and Goldstine's 80-page paper was the first to carry out a thorough and definitive error analysis for computer computations. Of interest to us are the first two chapters in which they discuss the sources of error in computer computations and, in particular, they identify a class of numerical data types that we recognise as perfectly formed *computer arithmetics*. Their computer arithmetics are data types based on fixed

precision representations of rational numbers, in a base with $s$ digits, and equipped with the operators

$$+, \ -, \ \times, \ \div$$





# NUMERICAL INVERTING OF MATRICES OF HIGH ORDER

JOHN VON NEUMANN AND H. H. GOLDSTINE

## ANALYTIC TABLE OF CONTENTS



Presented to the Society, September 5, 1947; received by the editors October 1, 1947. Published with the invited addresses for reasons of space and editorial convenience.



The 1947 study of matrix computation belonging to L. J. Comrie

Their definitions of the fixed precision representation are clear, precise and general:

A *digital number x* is an *s*-place, base β, digital aggregate with sign:

$$x = \varepsilon(\alpha_1, \ldots, \alpha_s)$$

where $\varepsilon = +1$ or $\varepsilon = -1$ and $\alpha_1, \ldots, \alpha_s \in \{0, 1, \ldots \beta-1\}$.

In Footnote 13 (p.1032), they notice that these arithmetics can be found in the computers they know:

> All existing machines (or almost all) are decimal, that is, have $\beta = 10$. With rare exceptions $s = 7$ to 10, for example, on the familiar "desk" machines
>
> s = 8 or 10. The "Mark I" computer at Harvard University has $s = 11$ or 23.
>
> Non-decimal machines of the future are likely to adhere, at least at first, to the same standard: for example, $\beta = 2$, $s = 30$ to 40.

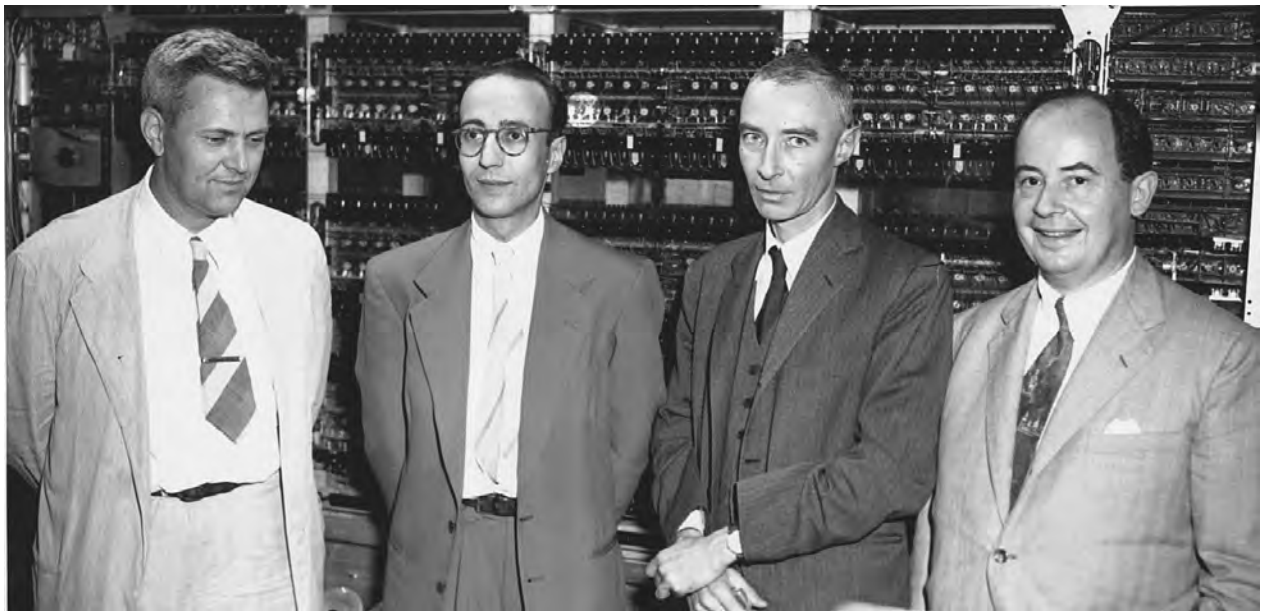
Von Neumann and Goldstine give neat error bounds for applying the four operations.

They also look at the fate of some of the classic laws that characterise arithmetic. They observe that addition does not present problems; that while commutativity of multiplication survives, the axioms of distribution, associativity of multiplication, and division as inverse to multiplication fail. However, they give nice error bounds to measure the failure of the axioms, such as:

$$|(a \times b) \div b - a| < |b|^{-1} \beta^{-1}$$

with a warning for |b| << 1. (See pages: 1038—1039) The remaining 5 chapters of the paper addresses matrices.

The shadow of the technical standard set by this analytical *tour de force* is interesting. It has been dispersed by focussing on stability (= sensitivity to errors in input), which proved to be more significant than quantitative error bounds. Alan Turing, in a paper the following year, also addressed matrix calculations in great generality, introducing LU factorisation and condition numbers. His colleague at the National Physical Laboratory, James H Wilkinson (1919—1986), later settled many of the problems in the course of his research (Wilkinson 1963, 1971). Also, working with floating point made error analysis easier. Although present in Konrad Zuse's designs, and in a sense a very ancient number representation, floating point made its presence felt in computing much later (Rojas 1997, Muller *et al* 2010).



*L-R: Julian Bigelow, Herman Goldstine, J. Robert Oppenheimer, John von Neumann*

Finally, let us remember that Herman Goldstine and John von Neumann's paper appeared after the 2nd World War had ended, after their work on the Manhattan Project (Figure 2), and as electronic computers – at least some of them – came into the light. Earlier that year, the first part of Goldstine and von Neumann's *Planning and Coding Problems for an Electronic Computing Instrument* was published at Princeton in April 1947; the remaining two parts appeared in April and August, 1948. These were important days for computer science. The works by Goldstine and von Neumann, including our numerical paper, count among the earliest attempts to make mathematical models of practical computing machines.

## van Wijngaarden on numerical analysis

The second offprint is by Adriaan 'Aad' van Wijngaarden (1916—1987) (*see overleaf)* :

> Numerical analysis as an independent science.
> BIT 6 (1966), 66 -81

and was mine from my time at the Mathematisch Centrum (MC), Amsterdam.

When von Neumann and Goldstine's paper appeared that year, in 1947, van Wijngaarden became head of computing at the MC. In the near 20 years that followed, he worked on building the first computer in the Netherlands, employing Edsger Dijkstra; supervising the theses of the rival pioneer machine designer Willem van de Pol and the pioneer semanticist Jaco de Bakker; and, of course, developing the language Algol 60, celebrated in this Newsletter in 2021. The intellectual world of Algol 60 is evident in the paper, which was presented earlier at a conference in 1964.

Now, Footnote 13 cited earlier suggests that von Neumann and Goldstine's mathematical model of computer arithmetics covered the computers of the period, at least as a far as they knew. So, its relevance to practical computation at that time is immediate. The scope of van Wijngaarden's paper is different and quite radical in its vision. It begins with the idea that numerical analysis and computer computation can be thought of as *independent* of the mathematics it is normally perceived to serve. Indeed, the numerical analyst:

> might also consider the result of the computation to be the thing that he wanted to have, and the "mathematical" concepts as approximations to his "numerical" concepts.

(Page 66). Computation uses rational numbers only; and rational numbers is the number system (= data type) needed to make measurements in the world and create the data that are the *raison d'être* for the scientific computers of the day.

The innovative idea of van Wijngaarden is to design an abstract, self-contained framework for numerical computation. It builds directly upon his intellectual experiences with Algol 60 with its aim to be a machine-independent language. He quotes at length the Algol 60 Report noting its silence on the type **real**. The paper is intimately connected with the Algol 60 world-view, but van Wijngaarden sees further.

His approach is to propose axioms for numerical computation that are independent of how the type **real** might be implemented, though they are shaped by computing with exact integers and floating-point numbers. He starts with the laws of equality and order, mindful of the fact that familiar (mathematical) properties fail. Next come the operations:

$$+,\ -,\ \times,\ \div$$

just as in von Neumann and Goldstine.

The errors that accrue are made explicit by axioms for a procedure *tolerance(x,e),* where *x* is an ideal value and *e* measures imprecision. The ideas and postulates gather pace as new operators are defined as little (Algol) procedures. Van Wijngaarden goes on to address summations, limits, continuity, integration and differentiation. He sensitively explores the immutable computational constraints of precision and finiteness.

The interplay of base functions and predicates and Algol procedures make the paper an abstract machine independent study of data types for the needs of numerical work.

Such an abstract view of computation was noticed but not always with understanding or admiration. William M Kahan (Turing Award 1989) is famed for his lifetime's work on the practical sciences of numerical computation. He is popularly associated with the IEEE Standard for Floating-Point Arithmetic (IEEE 754) of 1985. His experimental work and his technical reflections are deep.

In the early 1990s he wrote a critique of van Wijngaarden's then 20-year-old paper complaining of the complexity of its 32 axioms, and that it did not apply to the CDC 6600 – the supercomputer of the time. The last point displays a partial blindness to the nature of abstraction and portability in van Wijngaarden's project. These complaints were immediately rubbished by Edsger Dijkstra in EWD 1126-0.

Van Wijngaarden's paper is cited in Tony Hoare's An *axiomatic basis for computer programming* of 1969; Jaco de Bakker's early axiomatic work is also cited. Section 2 is called Computer Arithmetic and some axioms for the integers are presented, including ordering, addition, multiplication and 3 options for overflow. The axioms are needed to prove the correctness of a simple division program based on an iteration. Hoare's paper is cited for its proof rules for partial correctness tailored to the **while** programming language, and for the elegant case it makes for proving correctness. Its influence was immediate, though it was a while before the proof rules received a pukka logical analysis in Cook (1978). However, the true essence of this paper is in its title, of course: *axiomatisation!*

Rather than make a natural detour toward abstract data types in general, let me stick to computer arithmetics. There are several developments of so-called *exact computer arithmetics* all designed to implement very reliable exact computations. One example is *interval analysis* which focusses on implementations with rational number intervals; this approach can be reformulated in various ways, not least using ordered Scott-Ershov domains.

## Loose ends

Thus, abstract data types, their axiomatic specifications and associated term rewriting calculations and reasoning have a prehistory in arithmetical data types. But their subsequent mathematical theory, specification methods, conceptual spinouts, case studies, tools and applications have rarely returned to their ancestral numerical roots. Computer arithmetics rarely figure in the monographs, textbooks and papers of abstract data type theory.

So, what has abstract data type theory to say about *the* most important data type, the rational numbers?

A programme of research was started by Jan Bergstra and myself when we developed an equational specification of the rational numbers with the operations of $+, -, \times, ^{-1}$ in Bergstra and Tucker (2007). A division or inverse operator on the rationals is needed

to make an algebra that is generated by its constants – a condition called *minimality* that is necessary for an algebra to model a data type. But what to do about division by zero? Having little patience with partial operations in abstract data types and their painful semantics and logics, we took $0^{-1} = 0$.

This was the beginning of quite a journey for research into laws of computer arithmetic. We are focussed by different semantics for division by zero and the rational number data type. Semantic options for division by zero we have explored are

$$0^{-1} = \text{error}, \quad 0^{-1} = \infty \text{ (unsigned)}, \quad 0^{-1} = +\infty \text{ (signed)}.$$

In each case we have found equational specifications for corresponding algebras of rationals, and gone on to study the axiomatic classes to which they belong – a topic for another time (Bergstra and Tucker 2020, 2021). From this start, we hope to tie up some of the loose ends and shorten the distance from abstract data types to practical numerical computing.

## References

J. A. Bergstra and J. V. Tucker, The rational numbers as an abstract data type, *Journal ACM*, 54:2, Article 7 (April 2007) 25 pages. https://doi.org/10.1145/1219092.1219095

J. A. Bergstra and J. V. Tucker, The transrational numbers as an abstract data type. *Transmathematica* (2020). https://doi.org/10.36285/tm.47

J. A. Bergstra and J. V. Tucker, The wheel of rational numbers as an abstract data type, in Roggenbach (eds) *Recent Trends in Algebraic Development Techniques. 25th Workshop WADT, Lecture Notes in Computer Science 12669, Springer, 2021, 13 – 30.* https://doi.org/10.1007/978-3-030-73785-6_2

S. A. Cook, Soundness and completeness of an axiom system for program verification *SIAM J. Computing*., 7:1 (1978), 70 – 90. https://doi.org/10.1137/0207005

E. W. Dijkstra, *Who failed?* EWD 1126-0, Department of Computer Science, University of Texas at Austen, 1992. https://www.cs.utexas.edu/users/EWD/ewd11xx/EWD1126.PDF

C. A. R. Hoare, An axiomatic basis for computer programming, *Communications of the ACM*, 12:10 (1969), 576 – 580. https://doi.org/10.1145/363235.363259

J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhaüser, 2010.

J. von Neumann and H. Goldstine, Numerical inverting of matrices of high order, *Bulletin American Mathematical Society*, 53: 11, (1947), 1021 – 1099. https://projecteuclid.org/journals/bulletin-of-the-american-mathematical-society/volume-53/issue-11/Numerical-inverting-of-matrices-of-high-order/bams/1183511222.full

R. Rojas, Konrad Zuse's Legacy: The Architecture of the Z1 and Z3, *IEEE Annals of the History of Computing* 19:2 (1997) 5 – 16. https://doi.org/10.1109/85.586067

A.M. Turing. Rounding-off errors in matrix processes. *Quarterly Journal of Mechanics and Applied Mathematics*, 1:1 (1948), 287 – 308. https://doi.org/10.1093/qjmam/1.1.287

A. van Wijngaarden, Numerical analysis as an independent science. *BIT* 6 (1966), 66 – 81. https://doi.org/10.1007/BF01939551

J. H. Wilkinson, *Rounding Errors in Algebraic Processes*, National Physical Laboratory, HM Stationery Office, 1963.

J. H. Wilkinson, Modern Error Analysis, *SIAM Review*, 1971, 13:4: 548—568 https://doi.org/10.1137/1013095

# Logic and Machines: Turing Tradition at the Logic School of Münster

Egon Börger[1] and Rainer Glaschick[2]

[1] boerger@di.unipi.it
[2] rainer@glaschick-pb.de

**Abstract.** We describe the influence Turing's early work on the *Entscheidungsproblem* had in the *Schule von Münster* where it triggered the investigation of intimate links between logic and computation years before they became a main research theme in the science of computing. We report in particular on the work in the *Turingraum* where illustrative physical models of some outstanding (in particular universal) Turing and related machines were built some of which found their way into *teaching algorithmic thinking* in primary and secondary schools. We also point out how the investigation of various logical models of computation, initiated in Münster in the middle of the 1960s, led in the 1990s in a natural way to the practical but mathematically rigorous *Abstract State Machines Method* for a well-documented stepwise development of executable code by constructing logical models which successively introduce the necessary details to implement an initial rigorous requirements model.

## 1 Introduction

The story of this paper starts with two events in 1936 which involved Alan Turing and Heinrich Scholz.

**Fig. 1.** Heinrich Scholz and Alan Turing

**A.** In 1936 the Logic School of Münster (*Schule von Münster* [173, p.15], see also [219]) became institutionally visible, namely when **Heinrich Scholz**,

working since 1928 as professor of Philosophy at the University of Münster with a growing focus on mathematical logic and foundational issues, succeeded to establish as an independent part of the Philosophisches Seminar a "Logistische Abteilung" which eventually in 1950 became the *Institut für mathematische Logik und Grundlagenforschung* (Institute of Mathematical Logic and Foundational Research) at the mathematics department of the university.[1]

The group of 10 doctoral students who worked with Scholz between 1930 and 1953 in Münster distinguished itself by strong *interdisciplinary interests* and an *openness of mind towards applications of logic* in various fields, attracted and stimulated by the extraordinary wide range of interests and knowledge of its founder. These properties have been honored by the two followers as head of the school and director of the institute, namely **Hans Hermes** (1953-1966)[2] and **Dieter Rödding** (1966-1984, see [32]), as well as by Scholz' last student **Gisbert Hasenjaeger** who worked with Scholz as student, assistant and Dozent from 1945 to 1962 when he moved from Münster to the University of Bonn to establish there yet another successful Department of Logic and Basic Research which he directed until his retirement in 1984 (see [173, pp.261–262], [264] and https://rclab.de/hasenjaeger/publikationen_von_gisbert_hasenjaeger).

**Fig. 2.** Hans Hermes, Gisbert Hasenjaeger and Dieter Rödding

In this context also two later associates should be mentioned who contributed to the work of the school in set theory, proof theory and recursion theory resp. in model theory, namely **Wilhelm Ackermann**, who joined the group as honorary professor from 1953 to his death in 1962,[3] and **Wolfram Schwabhäuser**, who submitted his doctoral thesis in 1960 in Berlin [268], under the supervi-

[1] For the detailed historical development of the institute see [14].

[2] Hermes moved in 1966 to the University of Freiburg where he founded (and directed until his retirement in 1977) a new group of logic, sometimes referred to as the *Freiburger Logikschule*, which too excelled by a wide spectrum of scientific interests including applications of logic to computing. See [265].

[3] In computer science Ackermann is known mainly due to the scheme he invented to define recursive functions which 'grow faster than any primitive recursive function'

sion of Scholz' former student Karl Schröter, and his Habilitation in 1965 at the University of Münster, worked in 1965/66 with Tarski at the University of Berkeley, joined in 1966 Hasenjaeger's logic group in Bonn and since 1973 until his early death in 1985 worked as professor for theoretical computer science at the University of Stuttgart.

**Fig. 3.** Wilhelm Ackermann and Wolfram Schwabhäuser

**B.** In 1936 **Alan Turing** wrote (and published in [17]) his epochal paper where he

- defined a class of idealized machines, nowadays called *Turing machines* (TMs) which he argues to model *computability*, i.e. to capture every calculation, performed by a human using pencil and paper, 'which could naturally be regarded as computable' [17, p.230] (showing also that it is equivalent to Church's notion of *effective calculability* defined in [5]),
- proved that however some simple machine properties (e.g. whether a given machine once started will ever print some given symbol) cannot be computed by any Turing machine,
- derived from the preceding impossibility result a negative answer to Hilbert's question about the *Entscheidungsproblem* (decision problem of first order logic), as did Church in [4] using his concept of effective calculability,
- showed that there are so-called *universal* Turing machines, i.e. single (programmable) machines which can do ('simulate' in a precise mathematical sense) the work of any other Turing machine.

**C. The two events are intimately related** [127]: Scholz and his school promptly recognised the epochal significance of Turing's solution to Hilbert's fundamental *Entscheidungsproblem*, which at the time was considered to be 'the main problem of mathematical logic' [73, Ch.11,12]; they understood that there

and therefore are not primitive recursive [6]; see [150] for an account of his deep contributions to logic.

is an intimate connection between abstract machines and logic languages to describe the machine behaviour, between logical expressivity and computational machine power.

In fact, Scholz presented Turing's freshly published work in a seminar talk at the just founded logic institute, a seminar that has been called the world's first seminar on computer science [9].[4] Hans Hermes who at the time was a doctoral student at the institute[5] published right away in 1937 a first paper [142] where he builds upon Turing machines. When in 1947, after the war and after the Habilitation in mathematics at the University of Bonn, Hermes returned as Dozent to Münster he started regular lectures on the subject (since 1949) and in 1952 and 1954 wrote two other papers [144,145] where he adapted in a natural way Turing's definition of computation by a finiteness condition that better fits computing decision problems (instead of numbers). This led him directly to the halting problem. The definitions eventually went into his computability book of 1961 [147]. Together with Martin Davis' computability book of 1958 [210] these two early Turing machine based textbooks (which both went through various editions and had each a follower [185,211]) formed the mind of generations of logic and computer science students concerning the basic concepts of computability and undecidability.

### 1.1 Content of the paper

In this paper we illustrate how the early encounter between Turing and the Logic School of Münster triggered there for half a century (until Rödding's early death in 1984) a thorough investigation of the relations between logic and (in particular machine-based) concepts and methods of computation, years before they became a major theme for the new science of computing.[6]

- In Sect. 2 we report on the early Turing reception in Münster.
- In Sect. 3 we report on the work where the School of Münster enriched the classical recursion theory by machine-based classifications of recursive functions and by investigating links between machine computations and logical

[4] Note that *ibidem* (see also https://en.wikipedia.org/wiki/Heinrich_Scholz) it is also reported that Scholz was the only scientist worldwide outside the inner Cambridge circle who asked Turing for a reprint of his decision problem paper [17], by the way later also of [16]. How much Turing appreciated this interest can be seen from https://ivv5hpp.uni-muenster.de/u/cl/. The authors of [251] found letters of 1952/3 where Scholz tried (without success) to arrange for Turing a visit to Münster.

[5] His doctoral dissertation on "Eine Axiomatisierung der allgemeinen Mechanik" [143] was submitted there in 1938.

[6] In this paper we do not mention further the well-known research activities of members of the school in traditional areas of logic, in particular set theory, model theory, and proof theory. Nor do we mention further the numerous and intensive contacts the members of the institute maintained with their colleagues in other logic centers in Europe and the US.

and algorithmic decision problems, work which contributed to the emerging machine-based complexity theory in theoretical computer science.

- In Sect. 4 we report on Hasenjaeger's and Rödding's work to 'materialize' in hardware illustrative models of some outstanding computing machines, built in what in the institute was called the *Turingraum*. We explain also the effect this practical work had on the theoretical development of new models of computation. In Sect. 9 we provide the underlying technical details.
- In Sect. 5 we report on Rödding's and his students' work on modular decompositions of different kinds of automata which found various applications in other disciplines, among them an outstanding and rather intuitive dynamic way to teach fundamental mathematical and algorithmic concepts to students of elementary and high schools (see Sect. 6).
- In Sect. 7 we point to some alternative concepts of computation which have been developed in the School of Münster since the 1960s. They were about computing with non-numerical objects and structures, like terms, trees, graphs (networks of automata), topological structures, etc. Since the late 1980s this theme found a renewed interest in theoretical computer science and eventually led to the discovery of Abstract State Machines (ASMs). We shortly explain how this concept, in addition to its theoretical interest for logic and complexity theory, led in the 1990s to the development (far away from Münster) of the *ASM Method* which enhances the practice of rigorous software design and analysis.
- In Sect. 8 we summarize the institutional impact the Logic School of Münster (together with the School of Freiburg) had on the advancement of mathematical logic, in particular in Germany, by contributing to and taking inspiration from the practice and the theory of computing.
- In an appendix (Sect. 10) we show the genealogy of the School of Münster.

## 2 Early Turing Reception in Münster

The presentation of Turing's paper [17] to the logic seminar in Münster triggered three early publications that used Turing machines (in 1937, 1952 and 1954), written by Hermes who in 1937 was doctoral student of Scholz. In each of these papers which were addressed to a general educated public Hermes used Turing's definitions but adapted them to the theme each paper proposed to explain. The main modification with respect to Turing's original paper concerned the concept of computation or run of a Turing machine. Hermes tailored it by a finiteness condition (together with requiring runs to start with some input and to terminate with some output) that fits the definition of decision procedures, replacing the computations of the more complex circle-free machines Turing had tailored for computing infinite 0-1-sequences that represent real numbers in binary decimal notation.[7] The finiteness condition led Hermes in a natural way to the halting problem which is of lower arithmetical complexity than Turing's circle-freeness

[7] Turing notably used binary fractional numbers in a time when the rest of the world thought decimal, except two years later Zuse and four years later Atanasoff [20].

problem. The definitions went into the lectures Hermes delivered on the subject regularly since 1949 (see the early lecture notes [146] published in 1955 and the computability book [147]) and became part of common usage.

### 2.1 Turing Machines to Compute 'Definite' Predicates

In [142] Hermes saw a possibility to contribute to a rational discussion concerning the foundational question whether in mathematics only 'definite' predicates should be allowed, as claimed at the time by intuitionists and constructivists in opposition to the 'classical' understanding of mathematics. A predicate $P$ (over the natural numbers or over words of any given alphabet) was considered to be *definite* if one can indicate a general procedure to decide for each argument $x$ in a finite number of steps whether $P$ is true for $x$ or not.

Hermes used Turing machines to replace the intuitive understanding of 'a general decision procedure' by a precise mathematical concept. Therefore he proposed to define $P$ as definite if (our wording) there is a Turing machine $M$ such that for every argument $x$,

1. *$M$ started with input $x$*
2. eventually *yields a result*, namely *yes*/*no* if and only if $P(x)$ is true/false.

This became the standard definition of what nowadays we call a *decidable* predicate $P$ or also a predicate whose decision problem is *algorithmically solvable* (see for example [210, p.69, Def.2.1]). The two features 1. and 2. had to be integrated into the conceptual framework Turing tailored to compute numbers [17].

Ad 1. To start a computation not with the empty tape—as Turing requires for computing a number—but to 'supply the machine with a tape on the beginning of which is written some input' is used by Turing in two places (without naming it by a definition): for the construction of a universal machine [17, p.241] and for proving the unsolvability of the *Entscheidungsproblem* [17, p.259]. Hermes uses this input mechanism without changing the 0-1-representation of the input on the tape.

Ad 2. To yield a result necessitates some way to determine when the computation reaches a point where it is ready to provide a definite output (read: the answer whether $P$ holds for the input $x$ or not). Turing refers to such a feature (without defining it) where in an indirect proof he speaks about computations of the hypothetical machine $\mathcal{D}$ that decides circle-freeness and considers "when it has reached its verdict" [17, p.247] for the given input. Hermes makes this explicit by introducing a special symbol, say $H$, such that $M$, once started,

- after a finite number of steps will print the symbol $H$,
- up to this step has printed a 0-1-sequence that is interpreted to yield the result of the computation,
- from this moment on will never print any more any digit 0 or 1.

Note that the third condition—which guarantees that the "verdict" concerning the input question has been pronounced and will not change any more—turns such machines into circular (maybe not even halting) machines, exactly

those Turing was not interested in. We see here that the PhD student Hermes paid some attention to not change anything in Turing's definitions (neither of TM-programs nor of their computations) but to only adapt the *interpretation* of specific TM-runs to make them fit for 'deciding a property in finitely many steps', instead of computing an infinite 0-1-sequence. In Sect. 2.2 we will see that 15 years later, at the age of 40, Hermes replaces this veiled way to trick a Turing machine to serve as a decision procedure by a more streamlined and explicit definition of decidability of predicates and analogously computability of functions.

### 2.2 Entscheidungsmaschinen and Halting Problems

In his second Turing machine paper [144] Hermes simplified the stipulations made in [142] and made them more explicit, adding a termination convention to obtain a conceptually simple concept of what he called *Entscheidungsmaschine* (decision procedure): input/output Turing machines which, started with some input, terminate their computation after a finite number of steps and provide as result a yes/no answer to the input related question.

Technically this is achieved by adding to Turing machine programs at least one dedicated termination instruction $(q, a, b, move, halt)$ whose execution in state $q$ when reading $a$ leads the machine to the successor state *halt* in which the machine will stop (read: has in its program no quintuple $(halt, ....)$ to execute). Then one gets the definition of an *Entscheidungsmaschine* $E$ we all became used to and explained already above, i.e. $E$ decides $P$ if and only if for every $x$ the machine if started with input $x$ halts after a finite number of steps and its computation result is 1 (e.g. in its current working field) if and only if $P(x)$ is true, otherwise the result is 0.

The historically interesting fact one can observe here is that with such a simple halting convention (of a kind everywhere used today) Hermes proves in half a page, using a standard diagonalization argument, the undecidability of various forms of what nowadays is called the *halting problem*, thus considerably simplifying Turing's proof of the undecidability of the circularity property [17, pp.246–247]. The same year also Kleene [248, p.382] has proved the undecidability of a halting problem.

### 2.3 Universality of Programmable Computers

In his third Turing machine paper [145] Hermes sets out to show why programmable computers, as already available in 1954, are universal in the sense that one can construct a programmable computer such that for every computing machine its computational power is included in the computational power of that computer. This mathematical endeavor requires a mathematical definition of a) 'computing machines' and of b) the 'computational power' of programmable computers and computing machines.

To turn b) into a mathematical concept Hermes views (computations of) computing machines as computing functions over natural numbers (p.42)[8] so that their computational power can be defined by the class of number theoretic functions they can compute. For a) Hermes assumes Turing's thesis that every computable function is a Turing computable function, relying upon the accumulated evidence available already in 1954 that the numerous attempts to mathematically define computability in various ways eventually all led to Turing computable functions. Therefore to construct a universal programmable computer it suffices to construct an idealized computer which can be appropriately programmed to simulate every given Turing machine $M$; the idealization refers to the assumption that the computer has infinite memory (to store the program code for $M$ and to simulate the computation of $M$ for any input).

This is what Hermes does in [145], introducing a certain number of normalizations of Turing machines and their computations to simplify the construction of the *computer* which, if appropriately programmed, for any given Turing machine $M$ simulates the work of $M$. Some of these normalizations and variations thereof went into [147] and are common usage nowadays.


**Fig. 4.** The three Computability and Logic books by Hermes

1. The first simplification concerns the programs: instead of quintuples Hermes considers flowcharts of 5 basic component-TMs whose connection structure

[8] The underlying mathematical abstraction is that computing machines can handle arbitrarily large natural numbers, thus requiring that the memory is infinite. In [17, p.231] this appears in the form that a Turing machine "may also change the square which is being scanned, but only by shifting it one place to right or left", assuming that to the right and to the left of any square the mathematically idealized machine always finds another square. If one prefers to see memory as potentially infinite, there must be an operation that allows one to import any time new memory elements (enough pencil and paper for "a man in the process of computing"[17, 231]) from somewhere.

can easily be encoded by corresponding links (gotos) between subprograms of the *computer* program, one for each basic component machine. Obviously it is assumed that there is enough memory (unbounded register content) to store the given flowchart. For the connection in the flowchart visualization each component has exactly one entry, the Test component has two exits yes/no, all the other components have at most one exit:

- R-ight machine,
- L-eft machine,
- M-ark machine (prints $*$ in the working field),
- N-ull machine (prints blank in the working field),
- T-est machine (whether working field is marked or not)

2. Simplification of the tape: tape with a left end and infinite to the right, only two symbols $*$, *blank*.
3. The following normalizations concern the concept of computation.
   - Initialization: exactly one component is distinguished as the currently-to-be-executed one. The initial tape is $\bar{n}$ ($n$ a natural number) with the working position on the rightmost marked field; $\bar{n}$ is the tape beginning with $blank * \ldots *$ ($n$ occurrences of $*$) with completely blank rest of the right-infinite tape.
   - Stop criterion: after having executed a component without exit, the machine halts. The final tape is $\bar{y}$ where $y = f(n)$ and $f$ is the computed function. Only total functions are considered.

.

With these normalizations of TMs it is easy to encode an $M$-configuration into the memory of a programmable computer as follows:

- place the sum of all $2^k$ for all marked fields $k$ into a register, say $reg(tape)$,
- place the number of the current working field into a register, say $workPos$.

Thus it only remains to construct for each basic component machine $c$ a program $pgm(c)$ the *computer* will execute to simulate the behaviour of $c$, using a small number of memory locations.

**Remark**. Historically two facts are worth to be noticed:

- Hermes uses (in 1954) a number register for the encoding of the tape. The needed idealization with respect to physical computers is that such a register can hold arbitrarily large numbers.[9]
- It is only a small step from the flowcharts of 5 basic TM-components

$$R, L, M, N, T$$

to Rödding's structured programming Turing machines, called *Turing operators* [78]. The goto-structure of the flowchart is replaced by an algebraic

[9] Assuming that adding 1 to a register content $n$ yields a register content $n + 1$ corresponds to Turing's assumption that each move to the right or to the left of a current tape square finds yet another tape square.

program structure that is defined using concatenation and iteration; otherwise stated the test component $T$ is normalized to appear only at the beginning and end of an interated subprogram $M$ so that it has the form

$$(M)_*$$

denoting to iterate $M$ until the symbol $*$ appears in the working field. The same for $(M)_{blank}$. These operators (and even more their register operator relatives, see Sect. 3.2) simplify a lot the construction of Turing or register machines to compute recursive functions [78,81,30].

## 3 From Recursion Theory to Complexity Theory

In Münster, the path from recursion theory to complexity theory has been opened by two lines of research which have been pursued already before theoretical computer science became an academic discipline:

- description of machine computations by logical formulae relating computation power of the machines (measured in terms of the structure and computational strength of elementary machine operations and of computation time and/or space) and logical expressivity of the formulae (Sect. 3.1),
- machine-based characterization of hierarchies of recursive functions relating computational means to mathematical expressivity (definability schemes classified mainly in terms of the nesting of forms of recursion in traditional equational definitions of functions) (Sect. 3.2).

### 3.1 Machine-Characterization of Logical Decision Problems

The study of logical and algorithmic decision problems was part of the logic curriculum and a major thread of logic research in Münster. The subject was treated in regular courses (see the lecture notes [146,79,80,87])[10], diploma (master) and doctoral theses [175,176,86], articles [110,174,154,151,152,236,187,189,48,72] and monographies [7,44]. We refer here to just a few characteristic examples, for a complete Annotated Bibliography concerning the classical *Entscheidungsproblem* see [44].

Two lines of research characterized the traditional approach to the *Entscheidungsproblem*: develop algorithms for decidable cases [7] and reduce the general

[10] The institute had the tradition, initiated by Scholz (see [153, p.45]), to make lecture notes available for many courses. See for example [120,8]. The first author knows from the time when he was responsible for the library of the institute that this library contains many historically valuable lecture notes and also reprints of papers from famous logicians. The lecture notes often contained new research results that were not published elsewhere. For example, in [80] an elegant and simple proof for the sharp Kahr reduction class was developed, based upon a geometrical representation of computations of register machines with two registers in the Gaussian quadrant. This proof is used in [44, Ch.3.1]. Other examples are mentioned below. The lecture notes tradition has been maintained by Hasenjaeger also in Bonn, see for example [122].

problem to the decision problem of smaller and smaller classes of formulae (*reduction classes*) [180]. Following Büchi [171], stronger and stronger reduction classes were obtained by refining Turing's method to formalize the computations of machines (but also of other algorithmic systems or games) of a given class by logic formulae of a given class such that the machine behaviour of interest is equivalent to the decision (usually the satisfiability) problem of the corresponding logical formula. Since the 1980s, initiated in [157,115], this method is applied also to determine lower bounds for the complexity of decidable cases.

Establishing such relations between machine-computation-power and logical expressivity then became a major theme of automata and complexity theory in theoretical computer science. To mention just one outstanding example of a complexity variation of Turing's result: the NP-*completeness* of the satisfiability problem of propositional logic formulae in conjunctive normal form proved in [249] is a polynomial-time-restricted version of the $\Sigma_1$-*completeness* of the decision problem of predicate logic proved by [17] and in fact can be shown (see [33, Sect.2.3.1]) to be an instance of a general parameterized scheme (which has been developed in [29,28]) for logical implementations of machine programs.

The scheme relates algorithmic systems and their logical descriptions in such a way that numerous recursion and complexity theoretical properties are easily carried over from combinatorial to logical decision problems [27,26]. This includes the theoretically especially interesting case of Prolog programs where for one object—a Horn formula—different interpretations and their complexity properties are related, namely the computational properties of its program interpretation (e.g. Turing universality) and the logical properties of its purely logical interpretation (e.g. $\Sigma_1$-completeness of decision problems) [48,34]. The scheme can also be instantiated (see [44, Sect.2.2.2] for details) to prove Fagin's characterization of NP as a class of generalized first-order spectra [227], a result which marks the origin of Descriptive Complexity Theory where systematically logic languages are designed to capture computationally-defined complexity classes [218], a branch of finite model theory to which the **Logic School of Freiburg** contributed considerably [84] (see the lecturers from the *Coloquio sobre Logica Simbolica* at the Centro de Calculo de la Universidad Complutense in Madrid (19.02. – 21.02.1975) in Fig. 5 with Hermes and his former students and colleagues in Freiburg Dieter Ebbinghaus (Münster 1967) and Jörg Flum (Freiburg 1969)).

The very notion of spectra—for any given formula the set of the cardinalities of its finite models—and the question how to characterize them (called *Spektralproblem* and formulated in [159]) have been discovered by Hasenjaeger, Markwald and Scholz when they saw Trachtenbrot's undecidability result for finite satisfiability [259]. In 1971, applying Büchi's machine description technique from [171] to register machines (as done in Rödding's 1968 lecture [79]) and using the Rödding hierarchy $DSPACE(f_n)$ of functions—defined in [77] in terms of $n$-fold exponential time-bounded deterministic register machine computations, starting with the Grzegorczyk class $E_2$ and contained in and exhausting the


**Fig. 5.** J. Prida, H. Hermes, J. Flum, D. Ebbinghaus, E. Börger, unknown (Madrid 1975, names listed from right to left)

Grzegorczyk class $E_3$[11]—Rödding and Schwichtenberg [237] provided an elegant $n$-th order logical description of $n$-fold exponential time-bounded register machine computations with the result that n-th-order spectra form a strict subelementary hierarchy and exhaust the class of Kalmár-elementary sets. The result strengthened the early discovery, established using different (model-theoretic and number-theoretic) means by Asser [117] (as student of Schröter a scientific grand child of Scholz) and Mostowski [15], that first-order spectra reside between the classes $E_2$ and $E_3$ of the Grzegorczyk hierarchy of primitive recursive functions. The result rediscovered most of Bennett's unpublished thesis [170] which was based on a sophisticated analysis of the complexity of schemes to define number theoretic functions and relations and not on the computational complexity of machine programs that compute such functions.

Independently, in [179] a similar (but technically more involved) formalization of bounded computations of *non-deterministic* Turing machines by 'spectrum-automata' is developed which yields a characterization of first-order spectra by sets which are exponential-time acceptable by non-deterministic Turing machines, result proved independently also in [226]. Christen, using a sophisticated refinement of Rödding's computation time classification of subelementary functions [77] for register machines working over words, instead of numbers, has completed the Rödding-Schwichtenberg result in his doctoral thesis [64] to the final characterization of n-th order spectra as the class of $NTIME(f_n)$-sets of positive integers, where $NTIME(f)$ is the class of sets which are accepted by a

[11] For the Grzegorczyk hierarchy see [30, Ch.C].

non-deterministic Turing machine in time $f(c|x|)$ (for some constant $c$ and input length $|x|$) and the $n$-fold exponential functions $f_n$ are defined by $f_1(x) = 2^x$ and $f_{n+1}(x) = 2^{f_n(x)}$. Note that also the critical part of this characterization can be proved by a simple instantiation of the computation description scheme developed in [29] (see [44, pp.52–53] for the technical details).

Another line of research on the spectral representation of predicates appears in a series of papers by Deutsch (see the list in the Annotated Bibliography in [44]) where he uses normal forms of recursively enumerable sets he had developed in his doctoral thesis [212] to sharpen numerous reduction classes by restricting the interpretation of the unique occuring binary predicate symbol to an $\epsilon$-like relation over transitive sets.

Numerous interesting questions about spectra of (fragments of) first-order or higher-order logics are still today without answer. For a survey of the extensive later developments of the Spektralproblem in theoretical computer science and logic see the detailed and very informative survey [11].

### 3.2 Machine-Characterization of Recursive Functions

The second major thread of research in Münster which influenced later developments in complexity theory was the machine-based characterization of hierarchies of recursive functions, relating mathematical definability schemes (see [229,193]) to computation time and/or resource consumption needed to compute functions by restricted Turing-like machines, in particular (structured) register machines (which Rödding discovered in 1959/60 before they appeared in the two—for the study of decision problems most influential—papers [214,250], see Sect. 4 and [125,233,230]).

**Fig. 6.** László Kalmár and Dieter Rödding (Lecture in Münster, 1970)

Rödding's doctoral thesis [75] on Kalmár's class of elementary functions triggered much further work on this theme in Münster, in particular [83,231,77], the doctoral dissertations [161,158,94,139] and [162,163]. See also [82, Ch.V.5].

Putting together the work done in Münster (which includes Heinermann's investigation of recursion classes in his doctoral dissertation [263]) with [12,242,10,19] yields equivalent characterizations of the Grzegorczyk hierarchy of classes of primitive recursive functions by measuring the nesting of bounded recursion, the nesting of recursion, the growth bound by branches of the Ackermann function [6], the nesting of loops and the computation time bound of register machines to compute the functions (see the Grzegorczyk-hierarchy theorem in [82, Ch.V3] and [30, C.II.1]).

A side remark: computer scientists may be interested in the observation that the use of structured register machines (called register operators, with an analogous definition of Turing operators, see [78,233],[81, Ch.I.4] and their use in [30]) also allowed to sharpen the famous Böhm-Jacopini characterization of computable functions (see [62]) which has played a role for the development of the concept of structured programming.[12]

The two books shown in Fig. 7 document with proofs, further references and detailed historical comments the way computing machine concepts led from recursion theory to complexity theory and from the study of highly unsolvable decision problems to the study of the exact complexity of their decidable subcases, development to which the Logic School of Münster contributed considerably, as has been outlined in this section.

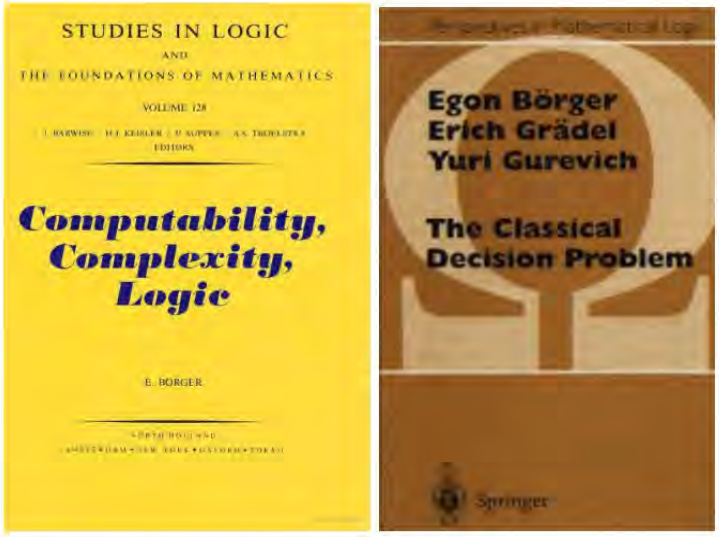



**Fig. 7.** Logic and Machines: From Decision Problems to Complexity

[12] For a generalization of the Boehm-Jacopini characterization from Turing machines to Abstract State Machines see [53, Proposition 5].

## 4 Turingraum

Copeland remarks in [71, p.54] that

> the mathematical representation of a Turing machine must not be confused with the thing that is represented – namely, an idealized physical machine

and concludes his chapter in the book by stating that

> What Turing described in 1936 was not an abstract mathematical notion, but a solid three dimensional machine containing (as he said) wheels, levers, and paper tape...

Turing was continuously engaged in practical projects (see Hodges [13]):

- in 1939, designed and built a mechanical machine to calculate roots of Riemann's zeta function (p.155),
- in 1944, invented and built a speech scrambler from valves (p.273),
- in 1945, designed the soft- and hardware for the Pilot ACE (p.333),
- in 1949, contributed a hardware random number generator for the Manchester Ferranti machine (p.402).

**Fig. 8.** Entrance to Turingraum in Münster (around 1962).

In the same spirit—combining theoretical work with the desire to build tangible theoretically well-founded machines—Hasenjaeger, soon joined by his doctoral student Rödding, pursued the goal to *materialize* (as Hasenjaeger wrote in

[125, p.182]) the theoretical concept of a universal Turing machine by some real physical machines one could also use to demonstrate the concept in lectures on the subject.[13] In the late 1950s Hasenjaeger and Rödding, supported by Hermes,[14] transfered these activities from Hasenjaeger's home to a dedicated room in the logic institute that was named *Turingraum* (Fig.8).[15] Here running physical models of small though computationally universal machines were built with simplest means until Rödding's early death in 1984; the same year Hasenjaeger retired but continued his materialization work at home.[16]

This materialization endeavor was triggered by a talk Friedrich Bauer gave in the middle of the 1950s at the logic institute in Münster where he presented his electro-mechanical model *Stanislaus* to evaluate algebraic terms in parenthesis-free notation. Upon Hermes' suggestion Hasenjaeger constructed for use in the institute a specimen of that machine, for details see Sect. 9.1.

This work soon led to the idea to build a physically running small universal Turing machine. As Hasenjaeger points out in [125] the "bottleneck was the materialization ... of a TURING tape" that was rewriteable.[17] The starting idea was to exploit the following ideas by Moore and Wang, of which Hasenjaeger only mentioned the first two ones:

- the reduction of the number of states in Moore's universal machine obtained by introducing a separate program tape plus an additional auxiliary tape (3-tape machine with only 15 states and 2 symbols [216]),
- Wang's observation [165] that erasing (overwriting) is not necessary; read and write-on-blank operations suffice (together with the move operations to the right/left) for a universal machine.
- Wang's proposal to use instructions instead of encoded state tables, which—at least in hindsight—was another step towards small and quick state machines.

[13] In [123, Part 1] Hasenjaeger speaks about physical 'models whose behaviour can be followed in "human" dimensions'.

[14] Already in his 1952 paper Hermes invites the technically interested reader to think about "how one can realize a Turing machine in practice"[144, footnote 5, p.185].

[15] The enlarged part below the room number shows that Turing's name was used as room name. The *Turingraum* together with a *Fregeraum* (reserved for the work on the Frege edition [156,116]) and two rooms for guest researchers formed a Dependance of the logic institute [267], given the lack of space in the castle which hosted the mathematical institutes until the end of the 1960s; at that time the *Turingraum* moved together with the Institute for Mathematical Logic to the new math building in the Einsteinstrasse.

[16] After Rödding's death the *Turingraum* has been disbanded by the new direction of the institute, the material was abandoned without further notice in a lumber room. To save it from greater damage it was quickly brought to Walburga Rödding (see [267]) who preserved it over the time, until in 2011, she and Hasenjaeger's family donated all physical artefacts from Hasenjaeger's legacy to the Heinz Nixdorf MuseumsForum in Paderborn (`http://www.hnf.de/`).

[17] Nota bene that Turing's paper suggests to use two tapes (even and odd fields) and to distinguish between erase and overwrite operations.

Out of the many artefacts that survived, the second author has been able to reconstruct Hasenjaeger's Mini-Wang machine, a Universal Turing Machine with only 4 states, 2 symbols and 3 tapes: a read-only program tape, a non-erasable working tape and a counter tape that is used to implement instruction skips. For the technical and historical details of this and related Turingraum machines see Sect. 9. Here we notice only that the effort to build universal but operationally surprisingly simple and running physical machines by no means lacked its scientific output:

- The remarkably small and physically executable Mini-Wang turned out to be efficiently universal among the dozens of conceptual universal Turing machines in the literature (see [217]). With hindsight one can also say that the investigation of computationally universal and with respect to a variety of parameters 'small' machines made the role explicit that different data, operations and architectural features (besides input/output mechanisms and stop criteria) play for the realization of the notion of computation:[18] number of symbols, states, tapes, the data type of and operations on tape contents (e.g. counters, stacks, read-only tapes, tapes with multiple parallel reads, cyclic shift registers), other topological structures than tapes (see in particular the work of Ottmann, Priese and Kleine Büning on universal machines we discuss in Sect. 5), etc. Hasenjaeger's abstract definition of register components of a net of machines in [119] looks like a presentiment of some particular classes of Abstract State Machines we discuss in Sect. 7.
- Rödding was led by the Q-tapes (counter tape) used in the Wang machines to the invention of register machines before their appearance in [214] and [250].

In fact, from the very beginning Rödding (who had enrolled at the university of Münster in 1956) got involved in Hasenjaeger's work and the creation of the Turingraum as working place for the construction of illustrative running machine models. When Rödding saw the use of a counter tape in the Wang machine he had the idea that counters alone could suffice to compute every partial recursive function.[19] He worked this out and presented the result to Hermes' Logic Seminar in Münster (see [125, p.184]), namely the definition of register machines (later

[18] This is related to Kleene's normal form theorem [247] that there are primitiv recursive functions *in*, *out*, *step*, *stop* such that every partial recursive function $f$ has the iterative form $f(x) = out \circ (step)_{stop} \circ in(k, x)$ for some $k$, where $(step)_{stop}$ denotes the iteration of the *step* function until the *stop* criterion becomes true (see the proof in [30, p.41]). Bruno Buchberger characterized the four component functions whose composition $out \circ (step)_{stop} \circ in$ defines a Gödel numbering (of the $n$-ary partial recursive functions for some $n$). These characterizations (see [30, Sect.BIII3] for proofs and references) show that one can design universal machines whose iterative component functions are of any a priori given (whether low or high) complexity, independently of each other.

[19] During the demonstration of some Turingraum machines at the Drei-Generationen-Kolloquium in Münster (see[46]) Hasenjaeger told the first author that he had asked Rödding whether one can represent sequences of natural numbers on 0-1-tapes of a universal Turing machine cheaply, in such a way that only an a priori fixed number

published in [78,233] with new results on structured programming normal forms) and the proof that every $n$-ary partial recursive function can be computed by a register machine with $n + 2$ registers and with prime number encoding even with only 2 registers, well before these results appeared in the famous papers [214,250].

The register machine concept turned out to be rather useful (see [168] for its role to pave the way for the Abstract State Machines concept, see Sect. 7). Rödding himself and his students made heavy use of them for an analysis of the computational power of numerous combinatorial systems, of the complexity of decision problems, of recursive functions, etc., as described in Sect. 3,5,6. Hasenjaeger used the elegant register machine proof by Jones and Matijasevich [178] for the theorem on exponential diophantine representation of enumerable sets in connection with his universal Turing machines to obtain a simple exponential diophantine predicate that is universal for the recursively enumerable sets [124].

So not surprisingly also register machines were a Turingraum theme, together with other components of Rödding's automata networks described in Sect. 5. See also Hasenjaeger's 'materialization' of (a general scheme of) register machines using SIMULOG instead of an electro-mechanical model [123]. See in particular the register machine materializations performed at the university of Osnabrück (see [99,63]) where these models have been applied with success for teaching algorithmic thinking at primary and secondary schools (see Sect. 6).

## 5 Networks of Machines

In the late 1960s until his early death Rödding together with a group of students analysed construction principles for (finite as well as infinite) automata and developed a theory for the modular decomposition of sequential automata by networks over a few simple basic automata [254,222,255,223,186,258,191]. Rödding's register operators (see [78,233]) appear here as nets with a particular graphical structure that visualizes the structured programming control. The theory made its way into the two textbooks [30,183] shown in Fig. 9.

Applications were found not only in computation theory [241,188,166,245,105] and logic [235,256,182]—where the inclusion of register components among the basic automata permitted to represent functionals of finite type, resulting in novel characterizations of the partial recursive functions (for the type 0 case), of the combinators K and S, of recursors, etc.— but also in theoretical biology [200], economics and systems theory [238,239,201,240,203,177], fault-tolerant switching theory [208,199,59] and Didactics of Mathematics (see Sect. 6).

Defining analogous nets of Asynchronous Parallel Automata (APA nets) [60] led to interesting results about concurrency [205,207,206]. The Turing spirit is particularly present in the applications of the theory of automata networks to the

of 1's appear on the tape. The answer was yes by representing $n$ as distance of a unique occurrence of 1 to the left end of the infinite-to-the-right tape $0^n1000\ldots$, i.e. a register where move-to-the-right means $+1$ and move-to-the-left $-1$.


**Fig. 9.** Two Books with a Chapter on Rödding's Automata Nets

construction of small alternative models of universal (Turing complete) computational systems, e.g. asynchronous cellular spaces [198], multi-dimensional Turing machines [253,192],[183, Sect.14.7] and 2-dimensional Thue systems [202], see also [204] and the comparative analysis in [266].

Further references appear in the survey [234], in the cited papers and in the two textbooks of the years 1985 [30, Sect.CIV3-4] and 2000 [183, Sect.14.6-7].

## 6 Computational Networks in Didactics of Computing

Rödding's work with register operators and networks of automata has triggered two particularly interesting didactical applications that have been elaborated by Elmar Cohors-Fresenborg and his group at the university of Osnabrück for teaching computational concepts in primary and secondary schools.

**Introduction of $n$-ary functions by register machines.** The first of these two applications is based upon the discovery presented in [95] that the register machine model of computation can be used with success to introduce in school the mathematical concept of multi-variable functions. Based upon various teaching experiments this idea has been elaborated first for teaching to high-school students (see [97], a book that according to Wikipedia and [261, p.40] has influenced the construction of the Know-how computer `https://en.wikipedia.org/wiki/WDR_paper_computer`, see also [101,124]); further experiments showed that the method can be adapted for secondary [103] (age 12–13) and even late primary [102] (age 10) school level.

From the very beginning of this work various specimens of a dedicated register machine model have been built to visually illustrate the computations so that the students can play with the machines. One copy of the first of these register machines [63] was purchased in 1976 for the Turingraum. It has been used in lectures by Rödding and by Ottmann (in Karlsruhe [68]). The physics

laboratory of the University of Osnabrück developed later versions on the basis of microprocessors offering an output mechanism to an external TV screen; these machines have been used with success in numerous schools in Germany and are part of the mathematical didactics study program at the University of Cologne (see https://mathedidaktik.uni-koeln.de/mitarbeiterinnen/prof-dr-inge-schwank/forschungs-und-lehrprojekte). Since 1982 also a simulation on PCs is available (see https://mathedidaktik.uni-koeln.de/mitarbeiterinnen/prof-dr-inge-schwank/forschungs-und-lehrprojekte/registermaschine/english-englisch).

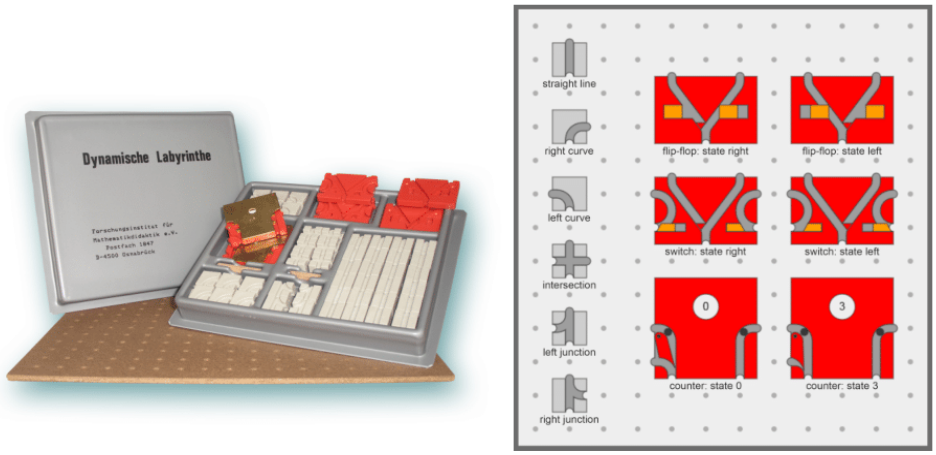



**Fig. 10.** The 9 Automata Construction Kit Bricks

**Automata construction kit for elementary schools.** In 1973 Elmar Cohors-Fresenborg started to exploit Rödding's networks of automata for didactical purposes (see [96]). The basic idea was to enable kids by a construction kit—consisting of bricks which are placed on a baseboard—to realize the computations of an automaton as walks through a net of basic components some of which perform a control action and others an operation on some data. For a simple to visualize but computationally universal concept of data, data operations and control, number register components (counters) came in handy with only two elementary operations +1, -1. Only two counters are needed; to realize their underlying Finite State Machine control they can be connected to an automata net of only two types of basic control components—flip-flop and switch—plus trivial support components like straight lines, curves, junctions, etc. (see [97]). The resulting *Automata Mazes*[20] construction kit (see Fig. 10,11) and its later software versions (see https://mathedidaktik.uni-koeln.de/mitarbeiterinnen/prof-dr-inge-schwank/forschungs-und-lehrprojekte/

[20] In German called *Dynamische Labyrinthe*, literally translated Dynamic Labyrinths, available via https://www.bildungsserver.de/onlineressource.html?onlineressourcen_id=10147.

automatentheorie-dynamische-labyrinthe) were applied with great success in elementary schools to teach algorithmic thinking in terms of counters plus typical railway net control! The offered teaching material that supports also self-study [100] has been translated to various languages including English, Chinese, and Dutch. It helps kids to realize algorithms as the result of non-verbal, action-oriented, motor thinking, an important form of mathematical reasoning pointed out already by van der Waerden in [262]. The didactical exploitation of Rödding's automata nets is applied with success also for very early training of particularly talented kids (see [246] and https://mathedidaktik.uni-koeln.de/fileadmin/home/ischwank/dynlab/maschinenintelligenz_mathematisieren_dynlab.pdf).

**Fig. 11.** flip-flop and switch bricks (in left position)

**Functional versus predicative thinking**. The two papers [98,69] report on extensive experiments with pupils who were observed during the construction of register machine programs or automata mazes. The result of these experiments led Inge Schwank to the conjecture that there are two forms of mental problem representation [244]: a *functional* one that is focussed on process runs, i.e. on the dynamic succession of process steps and their effect, and a *predicative* one where the attention is focussed on structural process elements and their relations. This difference has been experimentally confirmed by an analysis of brain currents and eye movements of test persons who were observed during a problem solving activity [215,104]; in [67] it is reported that these differences match the differences one can observe in how pupils approach algebraic or geometrical phenomena. In [70] Cohors-Fresenborg reports the rather interesting discovery of a similar difference in the mental representation managers have of business processes (see also [106,107]). It would be interesting to know whether Schwank's observation can be experimentally confirmed to also explain the two different system specification approaches advocated energetically by two camps of theoretical computer scientists, namely the declarative and the operational approach.

## 7 Computation on Structures Leading to ASMs

Since the middle of the 1960s various studies in Münster investigated computational concepts for objects which differ from numbers or words, using appropriate basic operations which work on those objects directly, without encoding. Early examples are the definition and characterization of primitive-recursive functions over hereditarily finite sets [76,232] and over sets of terms [111,112].[21] In the doctoral dissertation [94] register machines which operate on binary trees using a few natural basic operations are introduced to define and investigate the complexity of subelementary resp. elementary classes of functions over binary trees. The lecture notes [82] contain a chapter on generalizations of computable functions including a proposal for an axiomatic recursion theory along the lines of [109].

Later also Rödding's theory of networks of automata described in Sect. 5 led in a natural way to alternative computation models of topologically arranged automata or substitution systems [202,253,257,186,192] and asynchronous cellular spaces [198].

During those years two other logicians proposed to study computations over arbitrary relational (also called Tarski) structures [108,141]. Only much later were register machines used which operate on real (instead of natural) numbers, on rings, fields [196] or on finite relations over a fixed universe [65, Sect.4].[22] The query computability concept resulting from the last cited work played a major role in database theory. See [168] for a detailed historical analysis.

In 1982 the question appears whether one can replace Turing machines by a computation model over structures that captures the complexity class PTIME [66]. Three years later this question is extended in [134] to whether a computation model over structures can be defined which captures *every* sequential computational device, thus generalizing Turing's Thesis. The real breakthrough came with the idea,[23] nota bene conceived by a logician in an attempt to solve an epistemological question, to define a) a small number of elementary operations one can apply in any structure and b) a few composition (read: program construction) schemes to create composite sets of those operations, such that they suffice to define ('simulate') for every given algorithm its behaviour directly in terms of runs of an appropriate composition of those operations in the algorithm's 'natural' Tarski structures. This has been achieved in [135] by a simple mathematical definition of an arguably comprehensive algorithmic language. It lifts Finite State Machines (FSMs) and Turing Machines (TMs), which work over words, to **Abstract State Machines (ASMs)** which work directly over

[21] Note that in 1965 Hermes defined a logic of terms [149]. Much later the decision problem of this (and the pure $\epsilon$-logic) has been in investigated in Freiburg [184], see [44, Ch.5.3].

[22] The query language definition in [65] is given in terms of variables $y_i$ (which can be viewed as registers, each containing a finite relation) and a few appropriate basic operations which can be viewed as operating on register contents.

[23] With hindsight one is tempted to say 'simple idea', but as often happens with scientific discoveries the simple thoughts are the more difficult ones to find.

structures of *whatever* type (read: signature), using as basic action only guarded *term* (NOT only variable!) assignments **if** *condition* **then** $f(t_1, \ldots, t_n) := t$ and as composition scheme bounded synchronous parallelism.[24]

This operational definition of machines with arbitrarily abstract 'actions' broke with the at that time still main-stream declarative thinking in logic and theoretical computer science, but to the first author who had studied Tarski's Wahrheitsbegriff paper [21]—reading material for the introductory predicate logic course in Münster—and has been formed in the machine-based tradition in Münster, it quickly became clear that this definition enables to **mathematically support abstraction the way it is needed and used in the practice of computing** by engineers of software-intensive systems. In fact, ASMs permit to rigorously define algorithmic systems as computational models at whatever desired level of abstraction and to mathematically relate runs of an abstract and a refined machine (read: a machine that implements abstractions by more details) for a verification or test of the correctness of this 'implementation step'. These—by their abstract nature virtual—machines are not axiomatized by logical formulae but are computational models which drive the stepwise execution of the guarded-action-rules that capture the intended dynamic system behavior ('an evolution of states').


**Fig. 12.** Three Books on the ASM Method

The idea to use ASMs for a fully documented modular design, analysis and implementation of software-intensive systems by

- starting with appropriate rigorous but abstract requirements models (called *ground model ASMs* [36]) one can inspect like pseudo- (but semantically well-

[24] Other constructs, like **forall**, **choose**, **Call**, **let**, and other composition schemes, like asynchronous parallelism, can be easily integrated where useful. The work with ASMs revealed that the language can easily be tailored to the needs of domain specific applications. Note that the terms $f(t_1, \ldots, t_n)$ in guarded assignments are a general form of 'array variables' where the indeces are not just numbers, but arbitrary terms with possibly updatable values.

defined) code for its *appropriateness* with respect to the usually informally presented original requirements, and
- adding the implementation details by stepwise *ASM refinements* one can submit to test suits and to mathematical analysis for a *correctness check* [37][25]

triggered the **development of the ASM method** [54,252,50] (see Fig. 12), which is well-founded by its roots in logic and contributes effectively to the practice of computing, notably providing a system description and documentation technique of the kind Harel and Pnueli asked for in 1985 [137, p.480]:

> A natural, comprehensive, and understandable description of the behavioural aspects of a system is a must in all stages of the systems development cycle, and, for that matter, after it is completed too.

During the decade 1988-1998 the ASM community worked to make the ASM method fit for serious practical applications in a variety of computer science fields. A commented ASM bibliography for this period [47] counted 128 scientific contributions, which five years later became more than four hundred (see the commented bibliography in [88,54]); for references and the developments since then see https://www.abz-conf.org/methods/asm.

The **comprehensiveness of the concept of ASMs** is nowadays supported also theoretically, namely by numerous forms of Turing-like theses one can prove from natural axioms for appropriate classes of ASMs, characterizing for example sequential algorithms [136] (including a proof of Turing's Thesis for computable functions over the natural numbers) and their extensions to synchronous parallel algorithms [22,23,113], concurrent algorithms [51], recursive algorithms [52], reflective algorithms [181], etc. This work suggests the development of a realistic (theoretically well-founded and in the practice of software engineering helpful) complexity theory which is based not any more on Turing-like machines but on machines working directly over structures, avoiding extraneous encodings, see the very interesting recent survey paper [243].

In this respect it is interesting to note that half a century ago, in [274, p.6], the engineer Zuse critically remarked that much of research in theoretical computer science at the time disregards the badly needed dialogue between theoreticians and practitioners and that a logical algorithmic language is needed which helps the practitioner, in other words which is useful to reliably design, construct and analyze implementations. This is what the ASM method achieves, exploiting the machine-based pseudo-code-like yet abstract and semantically well-founded language of ASMs to design and analyse dynamic system behaviour.

## 8 Institutional Impact of the School of Münster

The Logic School of Münster had a strong impact not only on the scientific progress of mathematical logic as described above, but also on the institutional

[25] The concept of ASM refinement is not declarative but supports the direct description of system dynamics at different levels of abstraction. See [54] for details.

development of the discipline at German universities, especially in computer science departments where many positions became available for researchers in computational logic. We mention a few examples of influential science management activities of members of the group (besides those mentioned already in the chapters above).

Hermes acted as co-founder of the *Archiv für Mathematische Logik und Grundlagen der Mathematik* (1950) (later renamed to **Archive for Mathematical Logic**), as founding member of the *Deutsche Vereinigung für mathematische Logik und für Grundlagenforschung der exakten Wissenschaften* (**DVMLG**) (1962) and for many years has been co-editor of the *Journal of Symbolic Logic*. Together with Kurt Schütte from Munich he created and organized for many years the influential *Hermes-Schütte-Tagung* for mathematical logic in Oberwolfach. Schröter, who from 1936 to 1948 worked as student [194], assistant[195] and Dozent in Münster, established in 1950 the Institute for Mathematical Logic at the (now called Humboldt-) University of Berlin and in 1955 founded together with Asser the *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* (in 1991 renamed to **Mathematical Logic Quarterly**), acting as its editor until 1977.


**Fig. 13.** Hilbert/Ackermann and Scholz/Hasenjaeger Books

Members of the group wrote **influential books and textbooks**, some of them translated to other languages, on the main areas of logic. To mention a few: edition of the Frege Nachlass [156,116] (work that had been started by Scholz and his student Bachmann in the middle of the 1930s, has been destroyed by the fire of the university library during the bombardment of Münster on March 25, 1945, and has been taken up again by Hermes in the 1960s), textbooks or book chapters on logic [73,155,160,121,148,30,190,85] and computability [147,185,97,30,183], monographies on various subjects, e.g. term logic [149], the

*Entscheidungsproblem* [7],[44][26] (see also the first systematic textbook treatment of the *Entscheidungsproblem* in [73, Ch.12] which covers the results known at that time), metamathematics of geometry [269], proof theory [164].

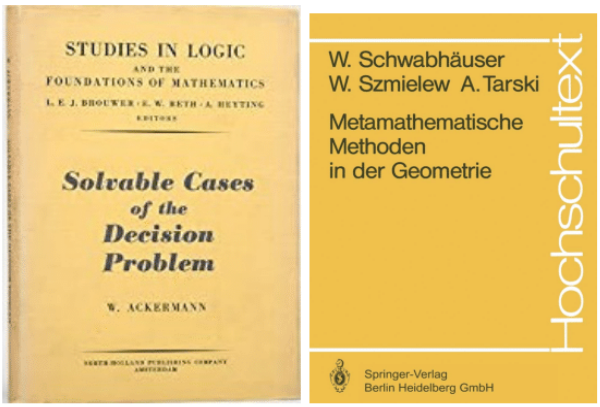



**Fig. 14.** Influential books by Ackermann and Schwabhäuser

Some numbers reveal the impact the computational focus of logic propagated by the *Schule von Münster* had on the **institutional growth** of the discipline in computer science departments of German universities. According to the Mathematics Genealogy Project [220] (see the list in the appendix)

- the school's first generation consists of 10 doctoral students,

[26] The first author has often been asked why it took [44] 25 years to appear. In 1972, on an invitation by the series editor Gert Müller, he had started to work on a volume in Springer's Lecture Notes in Mathematics with the material of his 1972/73 lectures in Münster on the *Classical Decision Problem* [87]. He stopped the project when in 1973 a doubt was expressed about a possible conflict with a book announced by Burton Dreben (Harvard); that book [74] came out in 1979 together with its companion book [197]. A critical analysis, performed with Yuri Gurevich during his visits to Münster and Dortmund (1978, 1983, 1985), of the complex machinery of Herbrand expansions used throughout in [74,197] and of the missing investigation of the algorithmic complexity of decidable cases eventually led to a new proposal of a comprehensive complexity account of the classical decision problem which was accepted by Gert Müller for Springer's Perspectives in Mathematical Logic, edited by the $\Omega$-group. After various tergiversations by Gurevich to find another author, eventually Börger wrote Part I (on undecidable classes), updated the annotated bibliography he had compiled in the 1970s and attracted Erich Grädel to join the project. Grädel's work on the complexity of decidable subclasses of logical theories in his doctoral dissertation [129,130,131] and his postdoc research in Pisa (Spring 1988 – Fall 1989) [132,133] made him the ideal person to complete the book by writing Part II (on decidable classes and their algorithmic complexity), except the Shelah class which has been written by Gurevich. The appendix (on tiling problems) has been written by C. Allauzen and B. Durand.

- its second generation consists of 81 doctoral students at the universities of Kiel, Berlin, Münster and Bonn (Bachmann 26, Schröter 16, Hermes 28, Hasenjaeger 11)
- D. Rödding, Scholz' second successor as director of the institute, supervised 15 doctoral students from 1966 until his early death in 1984.[27]

Alltogether in three generations Scholz has the extraordinary number of 1,625 descendants (Schröter 260, Bachmann 314, Hermes 392, Hasenjaeger 371, Rödding 246, Schwabhäuser 42).

*Logic and Machines* [46] (and more generally *Logic and Computation Theory* [31]) reflect the new horizon of scientific challenges the School of Münster discovered for the interaction of mathematical logic and computer science, in the spirit of the mathematician Turing and interestingly also of the engineer Zuse who submitted a manuscript [273] for a doctoral dissertation to Scholz.[28] *Logic and Machines* is the title of the Proceedings [46] of an **international symposium** which was organized in May 1983 in Münster—Hasenjaeger named it the *Drei-Generationen-Tagung*—to let reseachers come together who are interested in the cross-pollination between Logic and Computer Science. During this symposium Hasenjaeger and Rödding showed and explained to the participants some of their *Turingraum* machines.


**Fig. 15.** Books that Marked the Path to CSL

[27] See the list in [32] which corrects the one of the Mathematics Genealogy Project. It includes [169,198,177,59], see also https://www.uni-muenster.de/FB10/historie/anhangD.pdf. It mentions also five of Rödding's Diplom (Master) students who later have written a PhD thesis at other universities.

[28] Scholz wrote a positive evaluation, but due to the war and post-war conditions the PhD procedure did not reach its natural end. See [140, p.2]. Scholz showed great interest in Zuse's work; when Hans Lohmeyer, a former student of Scholz, worked with Zuse in Berlin he brought Scholz there for a visit (see [274, p.62]).

This was at a time when those who in Germany tried to bring logic and computing together experienced a strong resistence from a group of short-sighted professors of mathematical logic who did not understand the potential that computing held in store for their discipline.

The success of this symposium (with over 50 participants from 9 European countries and the US) gave the first author the idea to institutionalize such a forum by forming an annual **Computer Science Logic conference series**. After Rödding's unexpected death he attracted Hans Kleine Büning (one of Rödding's doctoral students) and Michael Richter (one of Hermes' early doctoral students in Freiburg) to join the endeavour. This was a year before the ACM/IEEE Symposium on *Logic in Computer Science* (LiCS, https://lics.siglog.org/) was launched in 1986. However, due to the adverse personal interest of somebody—who in 1985 after Rödding's death cut off the Turing tradition at the institute[29] (but obviously could not stop the strong development of computational logic in the scientific world) and triggered the first author's move to the University of Pisa—the conference series could not start in 1986 and not at the logic institute in Münster, but only a year later at the Institut für angewandte Informatik und Formale Beschreibungsverfahren in Karlsruhe.

The first seven years are documented in [40,41,42,43,39,38,56]. In 1992, during a Dagstuhl seminar [55] which has been attended by logicians and computer scientists from 14 countries the first author proposed to transform the CSL conference series into the annual gathering of a **European Association for Computer Science Logic** (EACSL), see [35] and for the complete list of CSL conferences and Proceedings https://dblp.uni-trier.de/db/conf/csl/index.html. Notably, when the EACSL upon Makowsky's proposal created an annual award for an outstanding dissertation in the area of logic in computer science, this distinction was named after Ackermann to stand for logic and computation. We quote from [18, p.VIII]:

> Together with ... LiCS, CSL counts as one of the most prestigious conferences in theoretical computer science focusing on the connections between logic and computing.

In this connection it is also interesting to remark that in the year 2000, a special conference subseries has been created that is devoted to "the foundational interconnections between *Logic and Computational Complexity*" (see https://www.cs.swansea.ac.uk/lcc/).

Also the development of the ASM method has been supported by a series of **International ASM Workshops**, co-founded in 1994 and until 2007 steered by the first author together with Yuri Gurevich. From the very beginning, many of these meetings were held as part of larger computer science conferences to easen the integration of the ASM method into current system engineering environments—ASM 1994 as part of the IFIP World Congress in Hamburg [221], ASM 1998 as part of the GI-Jahrestagung in Magdeburg [260], ASM 1999 as part of the Formal Methods Europe conference FME'99 in Toulouse, ASM 2001 as

[29] See the documentation in [25].


**Fig. 16.** Books that Marked the Start of ABZ Conferences

part of Eurocast'01 in Las Palmas [228,45]. Some more Proceedings have been published in [272,1,91,89,270,93,90,92].

During a Dagstuhl Seminar on rigorous methods for software construction and analysis [167] it became clear that state-based methods like ASMs, Z (Zermelo), B (Bourbaki) and others, all of which are based upon logic and set theory, have a lot of commonalities but also differences which should be clarified to enable practitioners to combine such approaches where this can help to develop and analyse reliable software for complex software-intensive systems. This led the first author to propose to Jean-Raymond Abrial, the creator of both the Z and the B (and Event-B) method and the leader of the community [2,3], to merge the ASM Workshops with the regular meetings of the Z and B user groups into a forum where common methods and ideas are investigated to reach a fruitful integration. This led to the establishment of the the biennual international **ABZ-Conference series** (`https://www.abz-conf.org/`) which has been launched in 2008 in London [49] and since then continues to be held regularly in Europe and Canada [49,213,172,271,24,61,209,225,224].

## 9 Analysis of Turingraum Artefacts

As already explained in Chap.4, Hasenjaeger had created several artefacts to show the *materialization* of theoretical concepts, and many of these still exist in various conditions by the effort of his family and W. Rödding. They are now

located in the Heinz Nixdorf MuseumsForum (HNF) in Paderborn, due to the initiative of its founding director Norbert Ryska. Whithout his engagement, in particular the *Mini-Wang* would still be unknown. Only the latter one has been analysed quite thoroughly and found to be still working; it is a universal Turing machine with (only) four states and three binary tapes, only one of them could be modified in a write-only-once manner. The other machines will be described here roughly; some of them might justify more deeper analysis. Also, for some artefacts their use is still unknown.

Documentation on these machines is often not existant; a few hints are in some of Hasenjaeger's publications. Several relevant texts appear in his paper legacy [118], but normally they are undated and difficult to match with the artefacts in the HNF.

The follwing comments on (some of) the artefacts now owned by the HNF are in hopefully historical order.

### 9.1 Kasimir

In his article [125] Hasenjaeger wrote that in 1956 F.L. Bauer from Munich reported in the *Logistisches Seminar* in Münster about an electromechanical model to evaluate parenthesis-free logical expressions, see [114]). His machine is in the *Deutsches Museum* in Munich, but currently not displayed.

Hans Hermes asked Hasenjaeger if he would like to build a similar machine ([125, p.182]):

> H. Hermes suggested that I should make a specimen of STANISLAUS for our institute, and F.L. BAUER sent me a blueprint of his version.

Bauer replied with the above mentioned blueprint and more information about his solution:

> Es sei:
> R die Anzahl von verschiedenen Variablen,
> S die Anzahl von verschiedenen logischen Operation einschlielich der Negation und der Identität, die Sie mit einem Formelrechner behandeln wollen.
> M die höchste Anzahl der in einer Formel vorkommenden Variablen und Operationszeichen.
> Dann benötigen Sie:
> 2 M Relais mit je einem Ruhe und Arbeitskontakt (für die Logik).
> 2 M Tastenstreifen mit je R+S Feldern, je Taste etwa 5 Ruhe und Arbeitskontakte;
> Einen doppelten Satz von je M Relais, deren Kontaktbestückung von 1 bis M/2 linear anwächst und ebenfalls linear abfällt (Für die Wegeschaltung)

This means, that the number of relays is quadratic with the number of terms. Hasenjaeger characterizes his machine KASIMIR in the above paper:

Its main features were:
(a) a three position switch for each place of the formula,
(b) a pair of relays to be activated by (a) for the shifts,
(c) a 10 position switch for each place of the formula, determining
(c1) the actual binary connective (ther are just 10 non-trivial ones) in connective-position of (a)
(c2) the subscript of the variable in variable-position of (a)
(d) two relays for each place of the formula for the actual connective
(e) but no realisation of a unary functor.
Being different from STANISLAUS our model needed a different name; I think `KASIMIR` was taken after K. AJDUKIEWICZ [1935].

Nothing had been published about *KASIMIR*, although it looks like the number of relays was only linear with the number of terms. It has not yet been studied if this is possible at all; so a complete reverse engineering of the machine might be valuable.

**Fig. 17.** Kasimir central part

Of the machine, two specimens of the first version survived (See Fig.17):

- one apparently complete machine (provided 2011 by W. Rödding)
- one badly preserved frame (found 2012 in Hasenjaeger's home in Plettenberg)

In Hasenjaeger's paper legacy [118, number 084] some notes may be found, at least for a later version (1977) with a different technology. A deeper analysis of the first version has been done by the second author.[30]

### 9.2 The "Alte Wang"

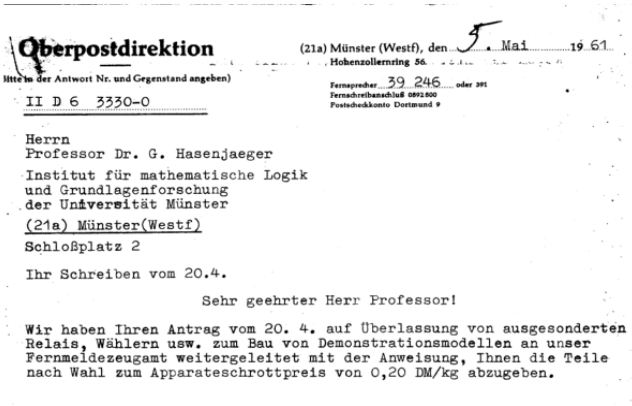
When Hasenjaeger learned about Moore's and Wang's proposals for practical Turing machines (see Chap.4), he started the work on his first Turing machine *materialization*.

This initiated the creation of a machine that could be programmed by short cables with 2.6mm plugs used in toy trains. It was originally called the *Wang*, and later *Alte Wang* (old Wang), see Fig.18.


**Fig. 18.** Alte Wang

Used were discarded relays[31] that were sold for their material price to educational institutions by the German post office, see Fig.19, which had a major office (*Oberpostdirektion*, OPD) in Münster.

[30] https://rclab.de/hasenjaeger/kasimir
[31] Flachrelais 28, i.e. flat relay, produced from 1928 on for the German post office



Oberpostdirektion

(21a) Münster (Westf), den 5. Mai 19 61

Hohenzollernring 56

(Bitte in der Antwort Nr. und Gegenstand angeben)

Fernsprecher 39 246 oder 391

Fernschreibanschluß 0892 800

Postscheckkonto Dortmund 9

II D 6 3330-0

Herrn
Professor Dr. G. Hasenjaeger
Institut für mathematische Logik
und Grundlagenforschung
der Universität Münster
(21a) Münster(Westf)
Schloßplatz 2

Ihr Schreiben vom 20.4.

Sehr geehrter Herr Professor!

Wir haben Ihren Antrag vom 20. 4. auf Überlassung von ausgesonderten Relais, Wählern usw. zum Bau von Demonstrationsmodellen an unser Fernmeldezeugamt weitergeleitet mit der Anweisung, Ihnen die Teile nach Wahl zum Apparateschrottpreis von 0,20 DM/kg abzugeben.

**Fig. 19.** OPD reply

As tapes, Hasenjaeger used 35mm perforated paper film used for contact prints.[32] Like ordinary 35mm film, it was perforated on both sides, so it could be split longitudinally, which Hasenjaeger did probably himself. [33] Hasenjaeger used paper film, as it was easier to punch than transparent celloloid film (used e.g. in Zuse's Z3), as the second authors experience in running the machine showed.

The tape drives were made in the workshop of the physics department in a quite professional way, see Fig.20.

In the middle is the slot through which the film guided, and at its bottom is the gear wheel that uses the perforation to move the tape many times without slack. On the left side is the stepper mechanism, that can move the tape left or right. On the right side is the sense and punch mechanmism. The punch is the outer tube, operated by the magnet with the heavy block on its armature.[34] To determine if there is a punched hole, a sense pin is moveable within the punch tube. It is normally kept away to allow free move of the tape. Activating the lower magnet, the sense pin is released and its position thereafter sensed by the leaf spring contact left to the magnet.

Hasenjaeger did not mention in [125] a very important feature of Wang's solution: Instead of encoding of a state table with an elaborate state machine to scan each line of the encoded table for a match, and then follow the actions, instructions are used like in stored progam computers, where the machine language is not a state table. Just the universal machine itself uses a state table, as in modern computers the microprogram that interpretes the instructions from

[32] At that time, often a (black-and-white) film was given to a drugstore to develop and produce 1:1 contact copies on this paper film, which did cost only a fraction of an enlarged copy of each picture.

[33] using a tool which is no longer known

[34] The obvious purpose is to provide enough mass to punch, even if such magnets have a much larger force short before closing anyhow.

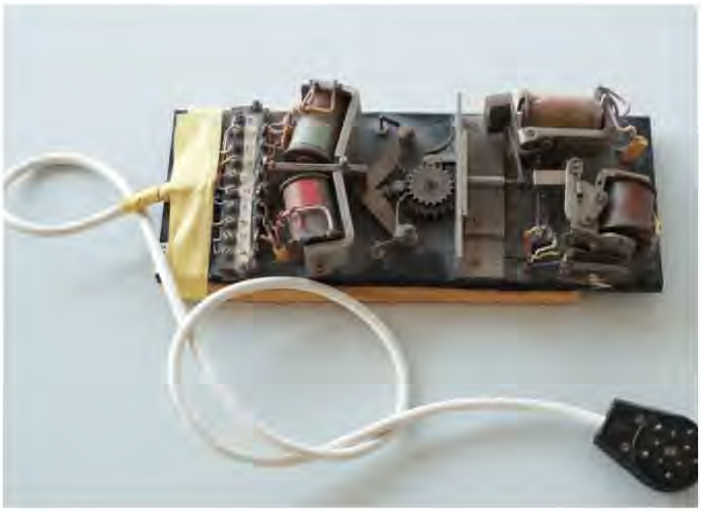


**Fig. 20.** Tape Drive *Alte Wang*

memory is close to a state table. This allowed a very compact program encoding (see example for the later *Mini-Wang* on page 39), otherwise such a machine would run too slow for even the smallest example.

Using the *Alte Wang*, Rödding had the insight that a single mark on an otherwise entirely blank tape could be used to store a number without modifying the tape at all, by just advancing the tape the given number of steps. Effectively, a number could be added and subtracted, but the total only read destructively. As Hasenjaeger wrote ([125, p.184]):

> But I think the fact that one counter ... suffices ... did suggest to D. Rödding to consider the possiblility of computing any recursive function by using only a bounded number of counting registers.

No consolidated documentation of this machine was found in Hasenjaeger's paper legacy [118], only several undated notes that could belong to the *Alte Wang*.

### 9.3 The Mini-Wang

The *Old Wang* was quite flexible, but bulky. This might have been Hasenjaeger's motivation to build another, smaller machine, with a fixed behaviour and as few states as possible in the state machine. He called this machine the *Mini-Wang*, see Fig.21.


**Fig. 21.** Mini-Wang

It is a universal machine with a central fixed state machine and three tapes, labelled P, Q and R:

- Tape `P`, containing program instructions
- Tape `Q`, a pure counter tape
- Tape `R`, the result tape (also for initial values)

Tape `P` is the program tape for the (encoded) instructions. It is a cyclic tape of 18 bits, using a selector switch[35] to sense 18 blue little DIP-switches that represent the program.

Tape `Q` is the skip counter for the program tape and equivalent to a (cyclic) tape with only one mark. Two selector switches are connected back-to-back, so that their equal position can be sensed modulo the number of positions. One is used for forward movement, the other one for backward movements.

Tape `R` (Fig.22) is the working and result tape. It uses the same 35mm halved contact sheet paper film as in the *Alte Wang*, but a different method to mark the tape. Instead of punching a (round) hole in the middle, a notch is punched at the upper end, and a lever is released vertically to sense the notch, Fig.23.

In contrast to the tapes of the *Alte Wang*, this tape drive was apparently made by Hasenjaeger himself. Note that the sprocket wheel comes from Meccano, sold in Germany by *Märklin*.

[35] also known as Strowger switch after its inventor

**Fig. 22.** Mini-Wang working tape

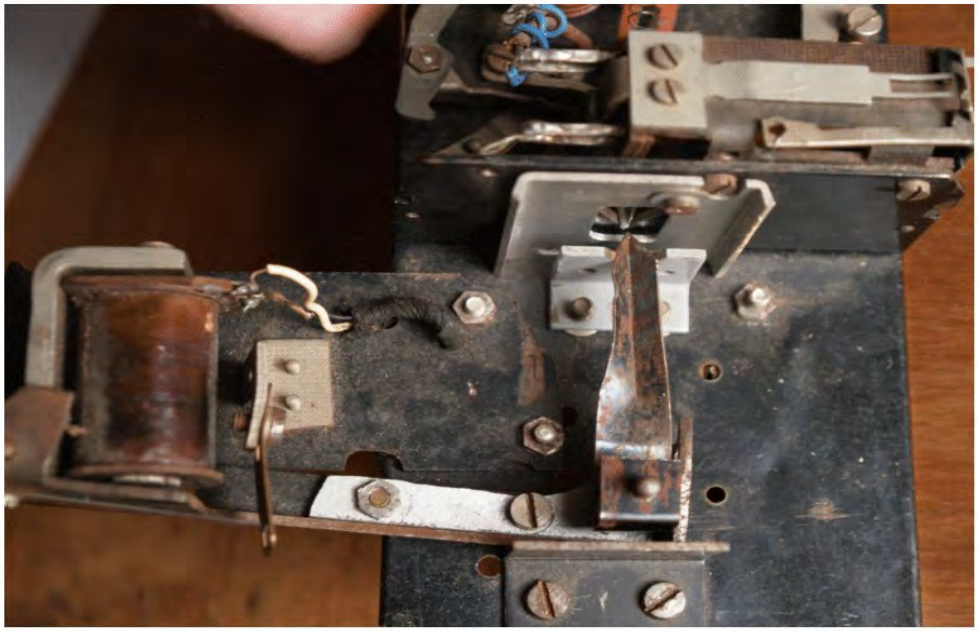


**Fig. 23.** Mini-Wang punch mechanism

The state machine uses 16 relays, which are not relays commonly used by the German post office.[36] The relays are properly orientated so that the gap between the contacts is vertical to avoid dust pile up.[37]

In Summer 2011, the second author reverse-engineered the machine to obtain the schematics shown in Fig.24. Four relays, labelled X, X', Y and Y' form two conventional flipflops for four states. Four relays are for the clock generator. The remaining 8 relays make up 4 Master-Slave flip-flops labelled R, L, O and U.

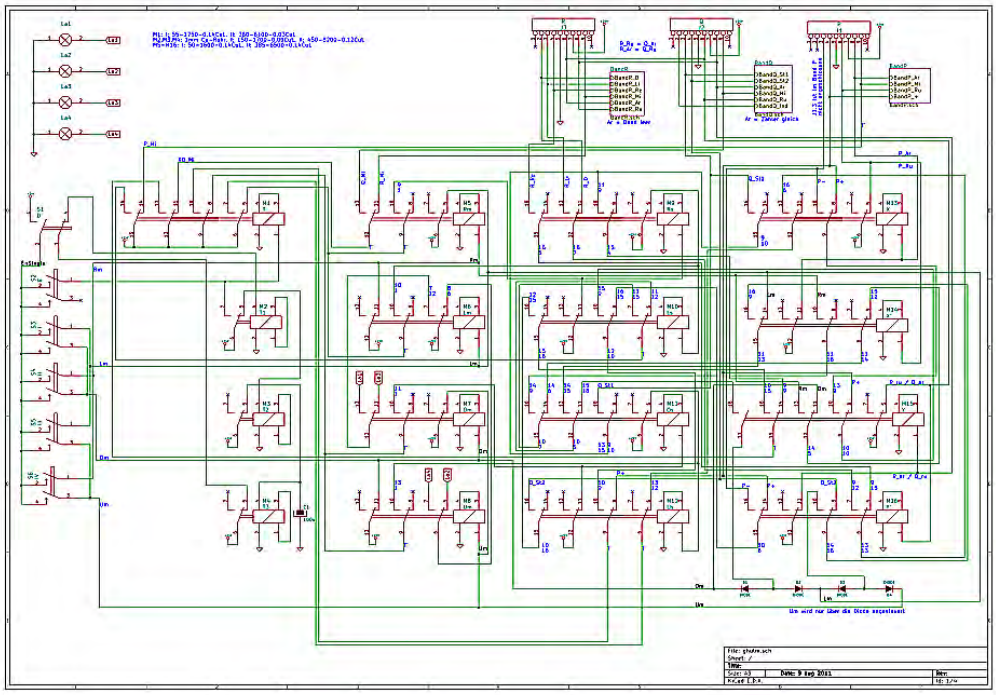


**Fig. 24.** Mini-Wang schematics

From the schematics, the state table was recovered, but found to be dubious, because the tape P would require repeats as forward jumps depending on the cyclicity of the tape.

[36] One reason might be that it is hard to find 16 relays with the same coils; in the second author's own collection of more than 200 post office relays, he was happy to find for a gray counter 5 such relays. Circuits were massively optimized for component count, ignoring the repair costs.

[37] This has been one of the major technological steps for larger reliability in German telephone exchanges using the Flachrelais 24. Zuse in his Z3 rebuild has the gap horizontally.

The recovered state table was:

```
Z       PQR     Z"      action

I       0._     II      P+
        0.*     II      P+
        1._     .       P+    M
        1.*     .       P+

II      0!.     III     P+ Q-
        00.     .       P+ Q+
        1!.     I       P+ Q- L
        10.     I       P+    R

III     0._     .       P+
        0.*     .       P+ Q+
        1._     I       P+
        1.*     IV      .  Q+

IV      0!.     .       P-
        00.     I       P+
        1!.     .       P- Q-
        10.     I       P+
```

As there was a unidirectional tape found with the others, in the action column there was originally no distintion between `P+` and `P-`. The instructions to be coded on tape `P` were

```
1       M   mark the tape
01      R   right move
001     L   left move
000<sup>n</sup>1   n   skip if the tape is marked
```

The conditional skip was originally assumed to be forward, using the cyclicity of the tape, which however made finding progammes rather difficult, as there are no neutral instructions (no-ops) to fill the tape.

The solution came when a bidirectional selector switch in its original boxing was found in Hasenjaeger's home in Plettenberg. Re-interpreting the schematics, it was clear that the machine was built for a bidirectional programme-tape, which was then created from the new switch and could be alternatively used, see Fig.25.

The Wang like encoding with variable length instructions requires in this case a minimum of 4 states to count the number of zeroes until it is clear that it is a skip. The `Q`-Tape may be used as a state extension and reduce the number of states to 3, which does not help to reduce the number of relays in practice.

A different punching tool was found and a small stock of already cut tape; both worked flawlessly at 24V, while using celloloid film required to increase the supply.

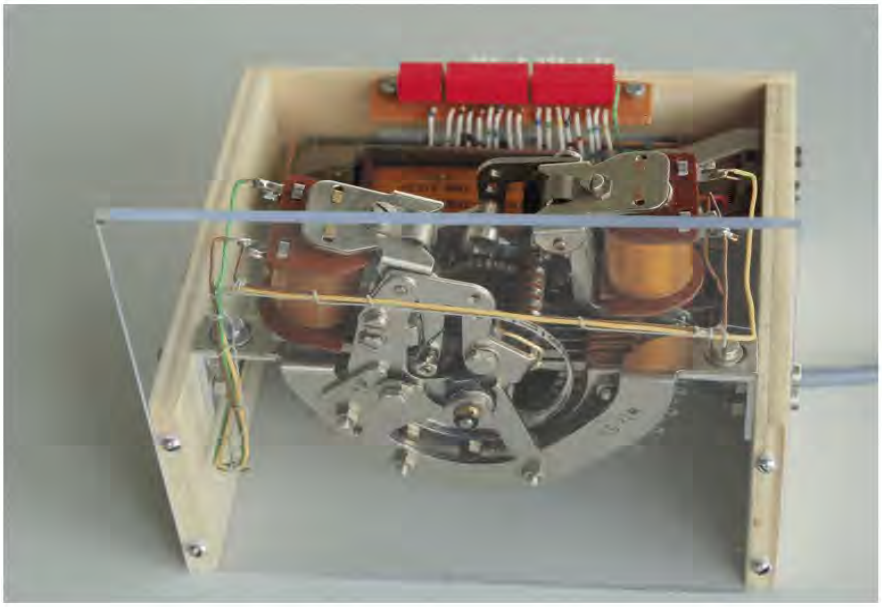


**Fig. 25.** Bidirectional progamme tape

So the above skip is finally a skip-back-if-marked, which corresponds to the repeat operator in Rödding's register machines, only the proper nesting is not enforced.

A programme to find the next space to the right and mark it (15 bits):

```
L   R  1     M 0
001 01 00001 1 0001
```

The first `L` is for the case that the tape is on a space. Then a tight loop moves right, checks for a mark, and if marked, repeats. If not marked, punches a mark, and a zero jump is a *dynamic stop*, which does jump infinitely, because the tape is marked.

More details are available. The formerly published descriptions (e.g. [127], [128]) are not up to date.[38]

To estimate the compactnesss of multi-tape Turing machines, the second author has proposed a Turing machine index ([126]) which is 24.0 (basic index) for this machine.

A proof that this machine is universal and also computationally efficient can be found in [217].

[38] Try [http://rclab.de/] for more details.

### 9.4 The RTL/70

A Box labelled *RTL/70* was found in his legacy in Plettenberg (see Fig.26), containing a base module and four pluggable modules, labelled `S`, `R`, `Q` and `P` Thus it looks like a 4-tape machine.

**Fig. 26.** RTL 70 in transport box and assembled

Used are early integrated digital circuits of the very early *Resistor-Transistor-Logic* (RTL) by Motorola ( `HEP570` to `HEP584`), for which not datasheets were found in the WEB.

An early attempt to reverse engineer the machine and re-build it using SMD replacements for the RTL ICs, was abandoned. Later, some documentation and the corresponding Motorola catalog was found, but the task was not restarted.

It might be suspected that this was a machine materializing the 2-state machine described in Hasenjaegers 1984 report ([138]), but that machine used a set of R-tapes, and this machine in its tailored box seems to be complete and intended for demonstrations.

The machine is very interesting because it is compact; its relevance can only be determined if analysed more extensively. The TM index cannot be guessed as the machine features are too vague.

### 9.5 The 1984 machine

Hasenjaeger gave the state table of a 2-state machine with 2 bits per instruction in [138]. It uses a `P`- `Q`- and a set of `R`-tapes, that could be cyclically selected, a technique he had persued since the *Alte Wang*.

The instructions on the P-tape are uniformly 2 bits long:

- E: enlarge (increment) the current R-tape
- D: decrement the current R-tape
- C: cyclically change to the next R-tape
- F: for *if*: conditional jump

As two F-instructions in a row are useless, the jump distance is encoded by the corresponding number of contiguous F-instructions.

The state table has been rearranged for better readability:

```
S PRJ   rj s    Remark
                Group 1: sequential instructions
0 c.0   #. .    cycle tape
  d10   -. .    decrement register
  e.0   +. .    increment register
  f.0   .. 1    start jumping
                Group 2: executing a jump
0 c.1   .- .    decrement J for c
  d.1   .- .    decrement J for d
  f.1   .. .    skip  f
                Group 3: do not jump as R is zero
1 c0.   .. 0    terminated by c
  d0.   .. 0    terminated by d
  f0.   .. .    ignore f
                Group 4: collect jump distance
1 c1.   .. 0    found c
  d1.   .. 0    found d
  f1.   .+ .    count number of 'f's
```

According to the text, the programme tape is assumed to by cyclic. As this is apparently a Turing machine simulating a register machine, backward jumps would be more appropriate, but require a different state table, that might nevertheless still have 2 states.

Until now, the only purpose of this machine seems to demonstrate the use of *Jones-Matiyasevich-Masking* to formally describe such a machine, which succeeded in an astonishing short proof.

The TM index is not as small as it seems, because the states of the tape multiplexer must be multiplied with the two visible states. The result tape is a single binary tape for the basic index, and there are 3+2 operations, thus the basic TM index is 6*sqrt(8*5/2) = 26.8. Penalties for the cyclic programme tape would be necessary. With backward jumps, there are two more actions in the state table, so the basic TM index is 6*sqrt(8*7/2) = 31.7, with no penalties.

### 9.6 The TTL machines

When TTL logic ICs became available, they were much cheaper than Motorola's RTL logic, and thus Hasenjaeger switched to this technology.

He created a large amount of modules that are connected using cables with 2mm plugs, see Fig.27.

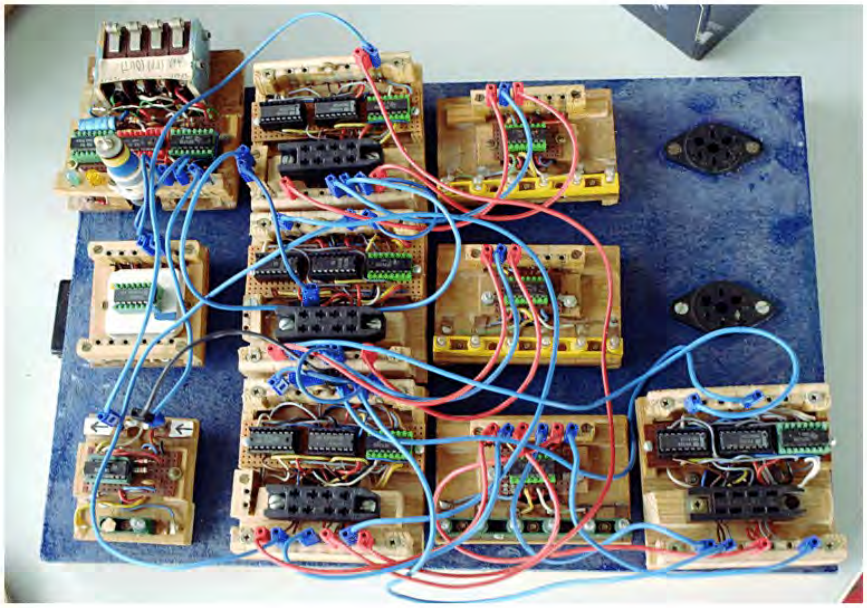


**Fig. 27.** TTL modules.jpg

Ground and power supply was from the bottom, thus the cables were only needed for signals.

No further analysis and inventory of the modules has been carried out so far.

### 9.7 More artefacts

A photo from the first *Turingraum* shows an object with old German telephone relays, (Fig. 28) which apparently is a shift register where the state is temporarily stored in capacitors in the front. Such shift registers could well serve as Turing tapes; if it is a ring counter, where only one relay is active ever, it is a Turing tape with just one mark, to be used as count register.

Similar artefacts are preserved but were not yet analyzed; some of these look like (another) relay shift register (see Fig. 29).

**Fig. 28.** Turingraum with relay shift register

**Fig. 29.** Relay shift register

**Acknowledgement.** We thank the following persons who have helped with criticism, suggestions, information, pictures: Volker Claus, Peter Päppinghaus, Andreas Podelski, Walburga Rödding, Uwe Schöning, Inge Schwank. We are particularly thankful to Elmar Cohors-Fresenborg who pointed us to most of the material in Sect. 6, and to Jonathan Bowen who after the presentation of his historical analysis of the community that has developed itself around the Abstract State Machines method (see [58,57]) suggested to write a companion paper that analyses the influence Turing's epochal 1937 paper had on the *Schule von Münster*, a community formed by activities that are focussed on the relations between logic and computing science.

## 10 Appendix: The Genealogy of the School of Münster

The Logic School of Münster in half a century had 65 doctoral students, listed below, and 1,353 descendants (data (not completely reliable) from [220], consulted on August 4 and November 25, 2021, with slight corrections due to direct knowledge of the first author, see in particular [32]).

- **Heinrich Scholz**' 11 doctoral students in Münster (1,265 descendants):

| Candidate | Year | Students | Descendants |
|---|---|---|---|
| Anna Holling | 1930 | | |
| Friedrich Bachmann | 1934 | 26 | 314 |
| Walter Kinder | 1935 | | |
| Hermann Schweitzer | 1935 | | |
| Eugen Roth | 1937 | | |
| Hans Hermes | 1938 | 28 | 398 |
| Shih-hua Hu | 1939 | | |
| Karl Schröter | 1941 | 16 | 263 |
| Eduard Arens | 1944 | | |
| Gisbert Hasenjaeger | 1950 | 11 | 378 |
| Werner Markwald | 1952 | | |

- **Hans Hermes**' 20 doctoral students in Münster.

| Candidate | Year | Students | Descendants |
|---|---|---|---|
| Heinz Gumin | 1954 | | |
| Arnold Oberschelp | 1957 | 7 | 20 |
| Walter Oberschelp | 1958 | 20 | 257 |
| Ludwig Brinkmann | 1961 | | |
| Horst Burwick | 1961 | | |
| Klemens Döpp | 1961 | 3 | 13 |
| Walther Heinermann | 1961 | 1 | 1 |
| Herbert Fiedler | 1962 | | |
| Dieter Titgemeyer | 1962 | | |
| Klaus Brockhaus | 1963 | | |
| Paul Röver | 1963 | | |
| Joachim Hornung | 1964 | | |
| Laurent Larouche | 1964 | | |
| Jürgen Genenz | 1965 | | |
| Friedrich-Karl Mahn | 1965 | | |
| Walburga Schwering | 1965 | | |
| Giorgio Germano | 1966 | 1 | 1 |
| Joachim Bammert | 1967 | | |
| Heinz-Dieter Ebbinghaus | 1967 | 7 | 47 |
| Klaus Rödding | 1967 | | |

- In Freiburg Hermes had the following 8 doctoral students: Reiner Durchholz (1968), Robert Kerkhoff (1968), Michael Richter (1968), Hubert Schwarz (1968), Jörg Flum (1969), Dieter Klemke (1970), Klaus Heidler (1973), Walther Kindt (1973).
  In total Hermes had 28 doctoral students and 398 descendants.
- **Gisbert Hasenjaeger**'s 11 doctoral students (378 descendants):

| Candidate | Year | Students | Descendants |
|---|---|---|---|
| Dieter Rödding | 1961 | 15 | 251 |
| Ronald Jensen | 1964 | 13 | 92 |
| Ulrich Perret | 1968 | | |
| Ibrahim Garro | 1972 | | |
| Wilhelm Johannes Backhausen | 1973 | | |
| Gerda Thieler-Mevissen | 1974 | | |
| Tassilo von der Twer | 1976 | | |
| Ralf Bülow | 1980 | | |
| Dimitrios Christodoulakis | 1980 | | |
| Peter Schroeder-Heister | 1981 | 6 | 22 |
| Emile Weydert | 1988 | | |

- **Dieter Rödding**'s 15 doctoral students (251 descendants):

| Candidate | Year | Students | Descendants |
|---|---|---|---|
| Helmut Schwichtenberg | 1968 | 16 | 25 |
| Michael Deutsch | 1968 | | |
| Thomas Ottmann | 1971 | 22 | 78 |
| Jürgen Bartnick | 1971 | | |
| Egon Börger | 1971 | 4 | 6 |
| Elmar Cohors-Fresenborg | 1971 | 11 | 14 |
| Hansjürgen Brämik | 1972 | | |
| Hans-Georg Carstens | 1972 | 22 | 58 |
| Helmut Müller | 1974 | | |
| Lutz Priese | 1974 | | |
| Peter Körber | 1976 | | |
| Hans Kleine Büning | 1977 | 9 | 13 |
| Klaus-Peter Kniza | 1980 | | |
| Joachim Müller | 1982 | | |
| Anne Brüggemann-Klein | 1985 | 2 | 2 |

## References


1. A.Blass, E.Börger, and Y.Gurevich, editors. *Theory and Application of Abstract State Machines*. Schloss Dagstuhl, 2002. Seminar Report 336. https://www.dagstuhl.de/02101.
2. J.-R. Abrial. *The B-Book*. Cambridge University Press, Cambridge, 1996.
3. J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, Cambridge, 2010.
4. A.Church. A note on the Entscheidungsproblem. *J. of Symbolic Logic*, 1:40–41, 1936.
5. A.Church. An unsolvable problem of elementary number theory. *American J. of Mathematics*, 58:345–363, 1936.
6. W. Ackermann. Zum Hilbertschen Aufbau der reellen Zahlen. *Mathematische Annalen*, 99:118–133, 1928.
7. W. Ackermann. *Solvable Cases of the Decision Problem*. North–Holland, 1954.
8. W. Ackermann. Von den natürlichen zu den reellen Zahlen. Lecture Notes, Institut für math. Logik und Grundlagenforschung, Winter Term 1961/2.
9. A.Clausing. Heinrich Scholz' early interest in Turing's papers. https://ivv5hpp.uni-muenster.de/u/cl/. Consulted July 11, 2021.
10. A.Cobham. The intrinsic difficulty of functions. In *Proc.1964 Congress for Logic, Mathematics, and Philosophy of Science*, pages 24–30, 1964.
11. A.Durand, D. Jones, J. Makowsky, and M. More. Fifty years of the spectrum problem: survey and new results. *Bull. Symbol. Logic*, 18:505–553, 2012.
12. A.Grzegorczyk. Some classes of recursive functions. *Rozprawy Matematiyczne IV*, IV:3–45, 1953.
13. A.Hodges. *Alan Turing: The Enigma*. Simon and Schuster, 1983.
14. H.-C. S. am Busch and K.F.Wehmeier. "Es ist die einzige Spur, die ich hinterlasse". Dokumente zur Entstehungsgeschichte des Instituts für Mathematische Logik und Grundlagenforschung. In H.-C. S. am Busch and K.F.Wehmeier, editors, *Heinrich Scholz. Logiker, Philosoph, Theologe*, pages 93–101. Mentis (Paderborn), 2005.
15. A.Mostowski. Concerning a problem of H. Scholz. *Zeitschr. f. math. Logik u. Grundlagen d. Math.*, 2:210–214, 1956.
16. A.M.Turing. Computing machinery and intelligence. *Mind*, LIX(236):433–460, 1937. https://doi.org/10.1093/mind/LIX.236.433.
17. A.M.Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937. https://doi.org/10.1112/plms/s2-42.1.230.
18. A.Raschke, E.Riccobene, and K.-D.Schewe, editors. *Logic, Computation and Rigorous Methods*, volume 12750 of *LNCS*. Springer-Verlag, 2021. Essays Dedicated to Egon Börger on the Occasion of His 75th Birthday.
19. A.R.Meyer and D.M.Ritchie. Computational complexity and program structure. IBM Watson Research Center at Yorktown Heights, Research Report RE–1817, p.1-15, 1967.
20. J. V. Atanasoff. Computing machine for the solution of large systems of linear algebraic equations. In B. Randell, editor, *The Origins of Digital Computers*, pages 305–325. Springer-Verlag, 1973.
21. A.Tarski. Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1:261–405, 1936.

22. A. Blass and Y. Gurevich. Abstract State Machines capture parallel algorithms. *ACM Trans. Computational Logic*, 4(4):578–651, 2003.
23. A. Blass and Y. Gurevich. Abstract State Machines capture parallel algorithms: Correction and extension. *ACM Transactions on Computation Logic*, 9(3), 2008.
24. F. Boniol, V. Wiels, Y.Ait-Ameur, and K.-D. Schewe, editors. *ABZ 2014: The Landing Gear Case Study*, volume 433 of *Communications in Computer and Information Science*. Springer, 2014.
25. E. Börger. Brief an G. Hasenjaeger 04.10.1985. Anlage: von M. Richter verfasste Dokumentation zur Rödding-Nachfolge. See Hasenjaeger Nachlass, Deutsches Museum München, NL 288 / 149.
26. E. Börger. On the construtions of simple first-order formulae without recursive models. In *Proc. Coloquio sobra logica simbolica*, pages 9–24, Madrid, 1975. Universidad Complutense.
27. E. Börger. A new general approach to the theory of the many-one equivalence of decision problems of algorithmic systems. volume 25, pages 135–162, 1979. Also published as vol.30 of R.Kaerkes and L.Merkwitz and W.Oberschelp: Schriften zur Informatik und Angewandten Mathematik, RWTH Aachen.
28. E. Börger. Decision Problems in Predicate Logic. In G.Lolli, G.Longo, and A.Marcja, editors, *Logic Colloquium'82*, pages 263–301. North-Holland, Studies in Logic and the Foundations of Mathematics vol.112, 1984.
29. E. Börger. Spektralproblem and completeness of logical decision problems. In E. Börger, G. Hasenjaeger, and D.Rödding, editors, *Logic and Machines: Decision Problems and Complexity*, pages 333–356. Springer LNCS 171, 1984.
30. E. Börger. *Berechenbarkeit,Komplexität,Logik*. Vieweg Verlag Braunschweig, 1985. 2nd ed.1986, 3d extended edition 1991, engl.translation Computability, Complexity, Logic (vol. 128 of Studies in Logic and the Foundations of Mathematics, North-Holland 1989), italian transl. Computabilità, Complessità, Logica vol.1: Teoria delle Computazione, Serie di Informatica, Bollati Borighieri 1989.
31. E. Börger, editor. *Computation Theory and Logic. In memory of Dieter Rödding*. Springer LNCS 270, 1987.
32. E. Börger. D.Rödding: Ein Nachruf. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 89:144–148, 1987.
33. E. Börger. Logic as Machine: Complexity relations between programs and formulae. In E. Börger, editor, *Trends in Theoretical Computer Science*, pages 59–94, 1988. A survey of the main results was presented to the centenary Scholz-Festkolloquium held at the logic institute in Münster on February 8–9, 1985.
34. E. Börger. Complexity of logical decision problems. In G. Corsi, M. Chiara, and G. Ghirardi, editors, *Bridging the Gap: Philosophy, Mathematics, and Physics*, pages 71–86. Kluwer Academic Publisher, 1993.
35. E. Börger. Ten years of CSL conferences (1987-1997). *EATCS Bulletin*, 63:61–63, 1997.
36. E. Börger. The ASM ground model method as a foundation of requirements engineering. In N.Dershowitz, editor, *Verification: Theory and Practice*, volume 2772 of *LNCS*, pages 145–160. Springer-Verlag, 2003.
37. E. Börger. The ASM refinement method. *Formal Aspects of Computing*, 15:237–257, 2003.
38. E. Börger, H. Büning, G.Jäger, S. Martini, and M.Richter, editors. *CSL'92*, volume 702 of *Lecture Notes in Computer Science*. Springer, 1993.
39. E. Börger, H. Büning, G.Jäger, and M.Richter, editors. *CSL'91*, volume 626 of *Lecture Notes in Computer Science*. Springer, 1992.

40. E. Börger, H. Büning, and M.Richter, editors. *CSL'87*, volume 329 of *Lecture Notes in Computer Science*. Springer, 1988.
41. E. Börger, H. Büning, and M.Richter, editors. *CSL'88*, volume 385 of *Lecture Notes in Computer Science*. Springer, 1989.
42. E. Börger, H. Büning, and M.Richter, editors. *CSL'89*, volume 440 of *Lecture Notes in Computer Science*. Springer, 1990.
43. E. Börger, H. Büning, M.Richter, and W.Schönfeld, editors. *CSL'90*, volume 533 of *Lecture Notes in Computer Science*. Springer, 1991.
44. E. Börger, E.Grädel, and Y.Gurevich. *The Classical Decision Problem*. Perspectives in Mathematical Logic. Springer-Verlag, 1997. Second printing in "Universitext", Springer-Verlag 2001.
45. E. Börger and U. Glässer. Abstract State Machines 2001: New developments and applications. In E. Börger and U. Glässer, editors, *J. Universal Computer Science*, volume 7(11), pages 914–917. Springer-Verlag, 2001. Selected extended papers from 8th international ASM workshop.
46. E. Börger, G. Hasenjaeger, and D.Rödding, editors. *Logic and Machines: Decision Problems and Complexity*, volume 171. Springer LNCS, 1984.
47. E. Börger and J. Huggins. Abstract State Machines 1988–1998: Commented ASM bibliography. *Bull. EATCS*, 64:105–127, 1998.
48. E. Börger and U. Löwen. Logical decision problems and complexity of logic programs. *Fundamenta Informaticae*, 10:1–34, 1987.
49. E. Börger, M.Butler, J. P.Bowen, and P.Boca, editors. *Abstract State Machines, B and Z*, volume 5238 of *Lecture Notes in Computer Science*. Springer, 2008. First International Conference ABZ 2008.
50. E. Börger and A. Raschke. *Modeling Companion for Software Practitioners*. Springer, 2018. ISBN 978-3-662-56641-1. For Corrigenda and lecture material on themes treated in the book see http://modelingbook.informatik.uni-ulm.de.
51. E. Börger and K.-D. Schewe. Concurrent Abstract State Machines. *Acta Informatica*, 53(5), 2016. http://link.springer.com/article/10.1007/s00236-015-0249-7, DOI 10.1007/s00236-015-0249-7. Listed as Notable Article in ACM 21th Annual BEST OF COMPUTING, see www.computingreviews.com/recommend/bestof/notableitems.cfm?bestYear=2016.
52. E. Börger and K.-D. Schewe. A behavioral theory of recursive algorithms. *Fundamenta Informaticae*, 177(1):1–37, 2020. DOI 10.3233/FI-2020-1915.
53. E. Börger and J. Schmid. Composition and submachine concepts for sequential ASMs. In P. Clote and H. Schwichtenberg, editors, *Computer Science Logic (Proceedings of CSL 2000)*, volume 1862 of *Lecture Notes in Computer Science*, pages 41–60. Springer-Verlag, 2000.
54. E. Börger and R. F. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer, 2003.
55. E. Börger, Y.Gurevich, H. K. Büning, and M.Richter. Computer Science Logic. 1992. Dagstuhl Seminar Report 40 (9229), https://www.dagstuhl.de/fileadmin/files/Reports/92/9229.pdf.
56. E. Börger, Y.Gurevich, and K.Meinke, editors. *CSL'93*, volume 832 of *Lecture Notes in Computer Science*. Springer, 1994.
57. J. P. Bowen. ABZ 2021 conference report. *FACS FACTS*, 2021(2):65–70, July 2021.
58. J. P. Bowen. Communities and ancestors associated with Egon Börger and ASM. In A. Raschke, E. Riccobene, and K.-D. Schewe, editors, *Logic, Computation and Rigorous Methods*, volume 12750 of *Lecture Notes in Computer Science*, pages 96–120. Springer, 2021.

59. A. Brüggemann. *Stochastische Zuverlässigkeit fehlertoleranter Netzwerke*. PhD thesis, Inst.math.Logik und Grundlagenforschung, Universität Münster, 1985. The scientific advisor of this dissertation has been D.Rödding, but the thesis was submitted after Rödding's death so that the Mathematics Genealogy Project does not associate it with D.Rödding.
60. A. Brüggemann, L.Priese, D.Rödding, and R.Schätz. Modular decomposition of automata. In E.Börger, G.Hasenjaeger, and D.Rödding, editors, *Logic and Machines: Decision Problems and Complexity*, pages 198–236. Springer Lecture Notes in Computer Science vol.171, 1984.
61. M. J. Butler, K.-D. Schewe, A. Mashkoor, and M. Biro, editors. *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 5th International Conference ABZ 2016*, volume 9675 of *Lecture Notes in Computer Science*, Linz (Austria), 2016. Springer.
62. C.Boehm and G.Jacopini. Flow diagrams, Turing machines and languages with only two formation rules. *Communications of the ACM*, 9:366–371, 1966.
63. C.Carstensen. Eine schaltalgebraische Realisierung von Registermaschinen. Prüfungsarbeit zur Ersten Staatsprüfung für das Lehramt an Realschulen, 1975. Pädagogische Hochschule Flensburg.
64. C.Christen. *Spektren und Klassen elementarer Funktionen*. PhD thesis, ETH Zürich, 1974.
65. A. K. Chandra and D.Harel. Computable queries for relational data bases. *J. Computer and System Sciences*, 21:156–178, 1980.
66. A. K. Chandra and D.Harel. Structure and complexity of relational queries. *J. Computer and System Sciences*, 25:99–128, 1982. https://doi.org/10.1016/0022-0000(82)90012-5.
67. C.Kaune. Das Wissen um Unterschiede in den kognitiven Strukturen von Schülerinnen und Schülern als Erklärung von Unterrichtsbeiträgen. *Zentralblatt für Didaktik der Mathematik*, 35:102–109, 2003.
68. E. Cohors-Fresenborg. Registermaschine. email of Nov 1 to Egon Börger.
69. E. Cohors-Fresenborg. On the representation of algorithmic concepts. In F.Lowenthal and F.Vandamme, editors, *Pragmatics and Education*, Boston (MA), 1986. Springer. https://doi.org/10.1007/978-1-4757-1574-3_13.
70. E. Cohors-Fresenborg. Individual differences in cognitive structures and the effect on business reengineering. In *Proceedings of the IV European Congress of Psychology*, pages 153–160, Göttingen, 1996.
71. J. Copeland. Turing's great invention: the universal computing machine. In J. Copeland, J. Bowen, M. Sprevak, and R. Wilson, editors, *The Turing Guide*, 2017. DOI:10.1093/oso/9780198747826.003.0013.
72. M. Deutsch. Ein neuer Beweis und eine Verschärfung für den Reduktionstyp $\forall\exists\forall^{\infty}(0,1)$ mit einer Anwendung auf die spektrale Darstellung von Prädikaten. *Zeitschrift für math. Logik und Grundlagen der Math.*, 38:559–564, 1992.
73. D.Hilbert and W.Ackermann. *Grundzüge der theoretischen Logik*. Springer, 1928,1938. English translation of the 2nd edition: Principles of Mathematical Logic, Chelsea Publishing Company, New York (1950).
74. B. Dreben and W. Goldfarb. *The decision problem: solvable cases of quantificational formulas*. Addison-Wesley, 1979.
75. D.Rödding. *Darstellungen der (im Kalmár-Csillagschen Sinne) elementaren Funktionen*. PhD thesis, Inst. für math. Logik und Grundlagenforschung, Universität Münster, 1961. Presented at the International Congress of Math., Stockholm 1962, and published in Arch. Math. Logik Grundlagenforsch. 7 (1965) 139–158.

76. D.Rödding. Theorie der Rekursivität über dem Bereich der endlichen Mengen von endlichem Rang. Habilitationsschrift, Institut für math. Logik und Grundlagenforschung, 1964.
77. D.Rödding. Klassen rekursiver Funktionen. In M. H. Löb, editor, *Proceedings of the Summer School in Logic (Leeds 1967)*, pages 159–222, 1968.
78. D.Rödding. Einführung in die Theorie der berechenbaren Funktionen. Lecture Notes (written by E.Börger), Institut für math. Logik und Grundlagenforschung, 1969. Reviewed in Mathematical Reviews (number 56 # 15384a/b).
79. D.Rödding. Höhere Prädikatenlogik: Interpolationstheorem, Reduktionstypen. Lecture Notes (written by H.Schwichtenberg), Institut für math. Logik und Grundlagenforschung, 1969.
80. D.Rödding. Reduktionstypen der Prädikatenlogik. Lecture Notes (written by E.Börger), Institut für math. Logik und Grundlagenforschung, 1970. Reviewed in Mathematical Reviews 57 (number 2903) and Zentralblatt für Mathematik 267 (number 02034).
81. D.Rödding. Einführung in die Theorie der berechenbaren Funktionen I. Lecture Notes (written by P. Koerber), Institut für math. Logik und Grundlagenforschung, 1972. Summer Term 1972.
82. D.Rödding. Einführung in die Theorie der berechenbaren Funktionen II. Lecture Notes (written by P. Päppinghaus), Institut für math. Logik und Grundlagenforschung, 1973. Winter Term 1972/73.
83. D.Rödding. Klasseneinteilungen im Bereich der rekursiven Funktionen. Lecture Notes, Institut für math. Logik und Grundlagenforschung, Winter term 1964/65.
84. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1995.
85. H.-D. Ebbinghaus, J. Flum, and W.Thomas. *Mathematical Logic*. Springer, 1995.
86. E.Börger. *Reduktionstypen in Krom- und Hornformeln*. PhD thesis, Institut für math. Logik und Grundlagenforschung, Universität Münster, 1971.
87. E.Börger. Reduktionstypen der klassischen Prädikatenlogik, Teil 1. Lecture Notes, Institut für math. Logik und Grundlagenforschung, 1972.
88. E.Börger. The origins and the development of the ASM method for high level system design and analysis. *J.Universal Computer Science*, 8:2–74, 2002.
89. E.Börger, editor. *Abstract State Machines and high-level system design and analysis*, volume 336 (2–3) of *Theoretical Computer Science (Special Issue)*. Elsevier, 2005. ISSN 0304–3975. Selection of extended papers from ASM'03 (Taormina, Sicily).
90. E.Börger, editor. *The Abstract State Machines method*, volume 77 of *Fundamenta Informaticae (Special Issue)*. IOS Press, 2007. ISSN 0169–2968. Selection of extended papers from ASM'05 (Paris).
91. E.Börger, A.Gargantini, and E.Riccobene, editors. *Abstract State Machines 2003. Advances in Theory and Practice*, volume 2589 of *LNCS*. Springer, 2003. Contains Proceedings of 10th ASM Workshop (Taormina, Italy). For a selection of extended workshop papers see [89].
92. E.Börger and A.Prinz, editors. *Quo vadis Abstract State Machines?*, volume 14 (12) of *J. Universal Computer Science (Special Issue)*. 2008. Selection of extended papers from ASM'07 (Grimstadt, Norway).
93. E.Börger, D. Beauquier, and A. Slissenko. Proc. 12th international workshop on Abstract State Machines ASM'05. Université Paris 12 (France), 2005. For a selection of extended workshop papers see [90].
94. E.Cohors-Fresenborg. *Subrekursive Funktionsklassen über binären Bäumen*. PhD thesis, Inst.math.Logik und Grundlagenforschung, Universität Münster, 1971.

95. E.Cohors-Fresenborg. Berechenbare Funktionen und Registermaschinen Ein Beitrag zur Behandlung des Funktionsbegriffs auf konstruktiver Grundlage. *Didaktik der Mathematik*, 3:187 –209, 1973.
96. E.Cohors-Fresenborg. Dynamische Labyrinthe. *Didaktik der Mathematik*, 1:1–21, 1976.
97. E.Cohors-Fresenborg. *Mathematik mit Kalkülen und Maschinen*. Vieweg, Braunschweig, 1977.
98. E.Cohors-Fresenborg. Verschiedene Repräsentationen algorithmischer Begriffe. *Journal für Mathematikdidaktik*, 6:187–209, 1985.
99. E.Cohors-Fresenborg and B.Reimers. Ein Demonstrationsmodell für Registermaschinen. *Der Mathematische und Naturwissenschaftliche Unterricht (MNU)*, XXVIII, 1975.
100. E.Cohors-Fresenborg, D.Finke, and S.Schütte. Dynamische Labyrinthe. *Osnabrücker Schriften zur Mathematik*, 1979. English version 11–19, Dutch version 31–39, also translated to Chinese and Indonesian.
101. E.Cohors-Fresenborg, M. Griep, and I. Schwank. Registermaschinen und Funktionen–Ein Schulbuch zur Einführung des Funktionsbegriffs auf der Grundlage von Algorithmen. *Osnabrücker Schriften zur Mathematik*, 22, 1979.
102. E.Cohors-Fresenborg and C. Kaune. *Von Anweisungen zu Funktionen*. Forschungsinstitut für Mathematikdidaktik e.V., Osnabrück, 2012. 3d revised edition.
103. E.Cohors-Fresenborg, C. Kaune, and M. Griep. *Einführung in die Computerwelt mit Registermaschinen*. Forschungsinstitut für Mathematikdidaktik e.V., Osnabrück, 1995.
104. E.Cohors-Fresenborg, S.Brinkschmidt, and S.Armbrust. Augenbewegungen als Spuren prädikativen oder funktionalen Denkens. *Zentralblatt für Didaktik der Mathematik*, 35:86–93, 2003.
105. E.Cohors-Fresenborg and I. Schwank. On the modelling of learning processes by $\alpha\beta\gamma$ - automata. In *Proc.7th International Congress of Logic, Methodology and Philosophy of Science*, pages 24–27, 1983.
106. E.Cohors-Fresenborg and I. Schwank. Kognitive Aspekte des Business Reengineering. *Gestalt Theory*, 18:233–256, 1996.
107. E.Cohors-Fresenborg and I. Schwank. Individual differences in the managerial mental representation of business processes. In R. P. et al., editor, *Managerial Behaviour and Business Processes: European Research Issues*, pages 93–106, Louvain, 1997.
108. E.Engeler. Algorithmic properties of structures. *Math. Systems Theory*, 1:183–195, 1967.
109. E.G.Wagner. Uniformly reflexive structures: on the nature of Gödelizations and relative computability. *Transac.American Mathematical Society*, 144:1–41, 1969.
110. T. Eichholz. Semantische Untersuchungen zur Entscheidbarkeit im Prädikatenkalkül mit Funktionsvariablen. *Archiv für math. Logik u. Grundlagenforschung*, pages 19–28, 1957.
111. F.-K.Mahn. *Über die Strukturunabhängigkeit des Begriffs der primitiv-rekursiven Funktionen*. PhD thesis, Institut für math. Logik und Grundlagenforschung, Universitä Münster, 1965.
112. F.-K.Mahn. Primitiv-rekursive Funktionen auf Termmengen. *Arch. math. Logik*, 12:54–65, 1969.
113. F. Ferrarotti, K.-D. Schewe, L. Tec, and Q. Wang. A new thesis concerning synchronised parallel computing – simplified parallel ASM thesis. *Theor. Comp. Sci.*, 649:25–53, 2016.

114. F.L.Bauer. The formula-controlled logical computer *Stanislaus*. *Mathematics of Computation*, 14:64–67, 1960.
115. M. Fürer. Alternation and the Ackermann case of the decision problem. *L' Enseignement Mathématique*, II:137–162, 1982.
116. G. Gabriel, H.Hermes, F. Kambartel, C.Thiel, and A.Veraart, editors. *Gottlob Frege. Wissenschaftlicher Briefwechsel*. Felix Meiner Verlag Hamburg, 1976.
117. G.Asser. Das Repräsentantenproblem im Prädikatenkalkül der ersten Stufe mit Identität. *Zeitschr. f. math. Logik u. Grundlagen d. Math.*, 1:252–263, 1955.
118. G.Hasenjaeger. Nachlass. Deutsches Museum München, Archiv, NL 288.
119. G.Hasenjaeger. Register-Maschinen. Deutsches Museum München, Archiv, NL 288/081. See [123].
120. G.Hasenjaeger. Einführung in die Mengenlehre. Lecture Notes (written by Hans-Rüdiger Wiehle) at University of Münster, 1953/4.
121. G.Hasenjaeger. *Einführung in die Grundbegriffe und Probleme der modernen Logik*. Alber, Freiburg and Munich, 1962. Engl.translation Introduction to the basic concepts and problems of modern logic, D. Reidel Publishing Company, Dordrecht, Holland, and Humanities Press, New York, 1972.
122. G.Hasenjaeger. Rekursive Funktionen. Lecture Notes (written by G. Seebach, Tassilo von der Twer and Jutta Klucken) at University of Bonn, 1971/2.
123. G.Hasenjaeger. Registermaschinen. *Contact*, 14 and 15, 1976.
124. G.Hasenjaeger. Zur Vor- und Frühgeschichte des (bis heute so genannten) "Know-How- Computers". pages 1–4, 1984. Apparently unpublished (but most of the material went into [125]). See Hasenjaeger Nachlass, Deutsches Museum München, NL 288 / 089.
125. G.Hasenjaeger. On the early history of register machines. In *Computation Theory and Logic*, volume 270 of *Lecture Notes in Computer Science*, pages 181 – 188. Springer, 1987.
126. R. Glaschick. A size index for multi tape Turing machines. Isaac Newton Institute Cambridge preprint, 18.7.2018. https://www.newton.ac.uk/files/preprints/ni12061\0.pdf.
127. R. Glaschick. Alan Turings Wirkung in Münster. *Mitteilungen der Deutschen Mathematiker-Vereinigung*, 20(1):42–48, 2012. https://doi.org/10.1515/dmvm-2012-0019.
128. R. Glaschick. Turing machines in Münster. In S. B. Cooper and J. van Leeuwen, editors, *Alan Turing: His Work and Impact*. Elsevier, 2013. ISBN 9780123869807.
129. E. Grädel. *The Complexity of Subclasses of Logical Theories*. PhD thesis, Universität Basel, 1987. For a summary see Bulletin of the EATCS vol. 34 (1988), 289–291.
130. E. Grädel. Subclasses of Presburger Arithmetic and the Polynomial-Time Hierarchy. *Theoretical Computer Science*, 56:289–301, 1988.
131. E. Grädel. Dominoes and the complexity of subclasses of logical theories. *Annals of Pure and Applied Logic*, 43:1–30, 1989.
132. E. Grädel. Size of models versus length of computations. On inseparability by nondeterministic time complexity classes. In *Proceedings of the Second Workshop on Computer Science Logic CSL 88, Duisburg 1988*, LNCS 385, pages 118–137. Springer, 1989.
133. E. Grädel. On logical descriptions of some concepts in structural complexity theory. In *Proceedings of the Third Workshop on Computer Science Logic CSL 89, Kaiserslautern 1989*, LNCS 440, pages 163–175. Springer, 1990.
134. Y. Gurevich. A new thesis. *Abstracts, American Mathematical Society*, 6(4):317, August 1985. abstract 85T-68-203.

135. Y. Gurevich. Evolving algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.
136. Y. Gurevich. Sequential Abstract State Machines capture sequential algorithms. *ACM Trans. Computational Logic*, 1(1):77–111, July 2000.
137. D. Harel and A. Pnueli. On the development of reactive systems. In K.Apt, editor, *Logics and models of concurrent systems*, pages 477–498. Springer-Verlag New York, 1985.
138. G. Hasenjaeger. Universal Turing machines (UTM) and Jones-Matiyasevich-masking. In E. Börger, G.Hasenjaeger, and D.Rödding, editors, *Logic and Machines: Decision Problems and Complexity*, volume 171 of *Lecture Notes in Computer Science*, pages 248–253. Springer Berlin / Heidelberg, 1984.
139. H.Brämik. *Beweistheoretische Charakterisierung der $\omega_3$-rekursiven Funktionen*. PhD thesis, Inst.math.Logik und Grundlagenforschung, Universität Münster, 1972.
140. H.Bruderer. Wie erfuhr die ETH Zürich von der Zusemaschine Z4? https://doi.org/10.3929/ethz-a-010001419, 2013.
141. H.Friedman. Algorithmic procedures, generalized Turing algorithms, and elementary recursion theory. volume 61 of *Studies in Logic and the Foundations of Mathematics*, pages 361 – 389. 1971.
142. H.Hermes. Definite Begriffe und berechenbare Zahlen. *Semesterberichte zur Pflege des Zusammenhangs von Universität und Schule aus den mathematischen Seminaren*, pages 110–123, 1937.
143. H.Hermes. *Eine Axiomatisierung der allgemeinen Mechanik*. PhD thesis, Universität Münster, 1938. Published in Leipzig as Heft 3 of Forschungen zur Logik und zur Grundlegung der exakten Wissenschaften.
144. H.Hermes. Maschinen zur Entscheidung von mathematischen Problemen. *Mathematisch-Physikalische Semesterberichte (Göttingen)*, pages 179–189, 1952.
145. H.Hermes. Die Universalität programmgesteuerter Rechenmaschinen. *Mathematisch-Physikalische Semesterberichte (Göttingen)*, pages 42–53, 1954.
146. H.Hermes. *Vorlesung über Entscheidungsprobleme in Mathematik und Logik*. Aschendorffsche Verlagsbuchhandlung, 1955. Vol.15 of Ausarbeitungen mathematischer und physikalischer Vorlesungen.
147. H.Hermes. *Aufzählbarkeit - Entscheidbarkeit - Berechenbarkeit. Einführung in die Theorie der rekursiven Funktionen*. Springer-Verlag, 1961. Various editions, english translation 1965, spanish translation INTRODUCCION A LA TEORIA DE LA COMPUTABILIDAD. See also the manuscript [146].
148. H.Hermes. *Einführung in die mathematische Logik - Klassische Prädikatenlogik*. Teubner Verlag, 1963. Second extended edition 1969, english translation Introduction to Mathematical Logic 1973.
149. H.Hermes. *Eine Termlogik mit Auswahloperator*. Springer-Verlag Berlin, 1965. Vol.6 of Lecture Notes in Mathematics. English translation Term Logic with Choice Operator in 1970.
150. H.Hermes. In memoriam WILHELM ACKERMANN 1896-1962. *Notre Dame Journal of Formal Logic*, VIII:1–8, 1967.
151. H.Hermes. Entscheidungsprobleme und Dominospiele. In K. Jakobs, editor, *Selecta Mathematica II*, pages 114–140. Springer, 1970.
152. H.Hermes. A simplified proof for the unsolvability of the decision problem in the case $\forall\exists\forall$. In R. Gandy and C. Yates, editors, *Selecta Mathematica II*, pages 307–310, Amsterdam, 1971. North-Holland.

153. H.Hermes. Logistik in Münster um die Mitte der Dreissiger Jahre. In H.Dollinger, editor, *Logik und Grundlagenforschung. Festkolloquium zum 100. Geburtstag von HEINRICH SCHOLZ*, volume 8 (Neue Folge), pages 41–52. Schriftenreihe der Westfälischen Wilhelms-Universität Münster, 1986.
154. H.Hermes and D.Rödding. A method for producing reduction types in the restricted lower predicate calculus. In *Formal Systems and Recursive Functions*, pages 42–47, Oxford, 1965.
155. H.Hermes and H.Scholz. Mathematische Logik. In *Enzyklopädie der math. Wissenschaften Vol. I, 1.1*, page 82, Leipzig, 1952. Teubner.
156. H.Hermes, F. Kambartel, and F. Kaulbach, editors. *Gottlob Frege. Nachgelassene Schriften*. Felix Meiner Verlag Hamburg, 1969.
157. H.Lewis. Complexity results for classes of quantificational formulas. *Journal of Computer and System Sciences*, 21:317–353, 1980.
158. H.Müller. *Klassifizierungen der primitiv-rekursiven Funktionen*. PhD thesis, Inst.math.Logik und Grundlagenforschung, Universität Münster, 1974.
159. H.Scholz. Ein ungelöstes Problem in der symbolischen Logik. *Journal of Symbolic Logic*, 17:160, 1952.
160. H.Scholz and G.Hasenjaeger. *Grundzüge der Mathematischen Logik*. Springer Verlag, 1961.
161. H.Schwichtenberg. *Eine Klassifikation der mehrfach-rekursiven Funktionen*. PhD thesis, Inst.math.Logik und Grundlagenforschung, Universität Münster, 1968.
162. H.Schwichtenberg. Rekursionszahlen und die Grzegorczyk-Hierarchie. *Archive f. math. Logik u. Grundlagenforschung*, 12:85–97, 1969.
163. H.Schwichtenberg. Eine Klassifikation der $\epsilon_0$-rekursiven Funktionen. *Zeitschrift f. math. Logik u. Grundlagen d. Math.*, 17:61–74, 1971.
164. H.Schwichtenberg and S.S.Wainer. *Proofs and Computations*. Cambridge University Press, 2011. ISBN: 9780521517690.
165. H.Wang. A variant of Turing's theory of computing machines. *J. ACM*, 4:63–92, 1957.
166. I.Schwank. *Präferenzgesteuerte $\alpha\beta\gamma$-Automaten*. PhD thesis, Universität Osnabrück, 1984.
167. J.-R.Abrial and U.Glässer. Rigorous Methods for Software Construction and Analysis. 2006. https://www.dagstuhl.de/06191. See Proceedings [**?**].
168. J.A.Makowsky. Some thoughts on computational models: from massive human computing to Abstract State Machines. In *Logic, Computation and Rigorous Methods. Essays Dedicated to Egon Börger on the Occasion of His 75th Birthday*, Springer Lecture Notes in Computer Science vol.12750, pages 173–186, 2021.
169. J.Bartnick. *Eine algebraisch-kombinatorische Darstellung der Prädikatenlogik*. PhD thesis, Institut für math. Logik und Grundlagenforschung, Universitä Münster, 1971. The scientific advisor of this dissertation has been D.Rödding, but the thesis is not listed in the Mathematics Genealogy Project [220].
170. J.Bennett. *On spectra*. PhD thesis, Princeton University, 1962.
171. J.Büchi. Turing machines and the Entscheidungsproblem. *Mathematische Annalen*, 148:201–213, 1962.
172. J.Derrick, J.Fitzgerald, S.Gnesi, S.Khurshid, M.Leuschel, S.Reeves, and E.Riccobene, editors. *Abstract State Machines, Alloy, B, VDM, and Z - Third International Conference ABZ 2012*, volume 7316 of *Lecture Notes in Computer Science*, Pisa (Italy), 2012. Springer.
173. J.Elstrodt and N. Schmitz. Geschichte der Mathematik an der Universität Münster. Teil II:1945–1969, Kap.7: Ehemalige Professoren 1945 - 1969. http://www.math.uni-muenster.de/historie/. Consulted July 11, 2021.

174. R. Jensen. Ein neuer Beweis für die Entscheidbarkeit des einstelligen Prädikatenkalküls mit Identität. *Archiv math. Logik u. Grundlagenforschung*, 7:128–138, 1962.
175. J.Genenz. Reduktionstheorie des Entscheidungsproblems im Prädikatenkalkül der ersten Stufe nach der Methode von Kahr-Moore-Wang, 1965. Diplomarbeit, Universität Münster.
176. J.Genenz. *Untersuchungen zum Entscheidungsproblem im Prädikatenkalkül der ersten Stufe*. PhD thesis, Inst.math.Logik und Grundlagenforschung, Universität Münster, 1965.
177. J.Müller. *Die mathematische Behandlung von Präferenz und Tausch unter Zugrundelegung des Automatenbegriffs*. PhD thesis, Inst.math.Logik und Grundlagenforschung, Universität Münster, 1982. The scientific advisor of this dissertation has been D.Rödding, but the thesis is not listed in the Mathematics Genealogy Project [220].
178. J. Jones and Yu.V.Matijasevich. Register machine proof of the theorem on exponential diophantine representation of enumerable sets. *Journal of Symbolic Logic*, 49:818–829, 1984.
179. N. Jones and A. Selman. Turing machines and the spectra of first-order formulas. *Journal of Symbolic Logic*, 39:139–150, 1974.
180. J.Suranyi. *Reduktionstheorie des Entscheidungsproblems im Prädikatenkalkül der ersten Stufe*. Verlag der Ungarischen Akademie der Wissenschaften (Budapest), 1959.
181. K.-D.Schewe and F.Ferrarotti. Behavioural theory of reflective algorithms I: Reflective sequential algorithms. arXiv:2001.01873. submitted 2020.
182. K.-P.Kniza. *Automaten und rekursive Funktionale endlichen Typs*. PhD thesis, Inst.math.Logik und Grundlagenforschung, Universität Münster, 1980.
183. K.Erk and L.Priese. *Theoretische Informatik. Eine umfassende Einführung*. Springer, 2000.
184. K.Heidler. *Untersuchungen zur Reduktionstheorie des Entscheidungsproblems in der Prädikaten- und Termlogik*. PhD thesis, Universität Freiburg, 1973.
185. K.Heidler, H.Hermes, and K.Mahn. *Rekursive Funktionen*. Bibliographisches Institut-Wissenschaftsverlag, Mannheim, Wien, Zürich, 1977.
186. H. Kleine Büning. *Über Probleme bei homogener Parkettierung von Z  Z durch Mealy-Automaten bei normierter Verwendung*. PhD thesis, Inst. für math. Logik und Grundlagenforschung, Universität Münster, 1977.
187. H. Kleine Büning. Some undecidable theories with monadic predicates and without equality. *Archiv math. Logik u. Grundlagenforschung*, 21:137–148, 1981.
188. H. Kleine Büning. Complexity of LOOP-problems in normed networks. In *Logic and Machines: Decision Problems and Complexity*, Springer LNCS 171, pages 254–269, 1984.
189. H. Kleine Büning and T. Lettmann. Classes of first order formulas under various satisfiability definitions. In *8th International Conference on Automated Deduction (CADE 1986)*, Springer LNCS 230, pages 553–563, 1968.
190. H. Kleine Büning and T. Lettmann. *Aussagenlogik: Deduktion und Algorithmen*. Teubner, 1994.
191. H. Kleine Büning and L.Priese. Universal asynchronous iterative arrays of Mealy automata. *Acta Informatica*, 13, 1980.
192. H. Kleine Büning and Th.Ottmann. Kleine universelle mehrdimensionale Turingmaschinen. *J. Inf. Process. Cybern.*, 13:179–201, 1977.
193. K.Rödding. *Zur Klasseneinteilung der rekursiven Funktionen nach Kleene*. PhD thesis, Inst. math. Logik und Grundlagenforschung, Universität Münster, 1967.

194. K.Schröter. *Ein allgemeiner Kalkülbegriff*. PhD thesis, Universität Münster, 1941.
195. K.Schröter. *Axiomatisierung der Fregeschen Aussagenkalküle*. PhD thesis, Universität Münster, 1943. Habilitationsschrift.
196. L.Blum, M.Shub, and S.Smale. On a theory of computation and complexity over the real numbers. *Bulletin American Math.Society*, 21:1–46, 1989.
197. H. Lewis. *Unsolvable Classes of Quantificational Formulas*. Addison-Wesley, 1979.
198. L.Priese. *Über einfache unentscheidbare Probleme: Computational- und constructional universelle asynchrone cellulare Räume*. PhD thesis, Inst. math. Logik und Grundlagenforschung, Universität Münster, 1974. The scientific advisor of this dissertation has been D.Rödding, but the thesis is not listed in the Mathematics Genealogy Project [220].
199. L.Priese. On the minimal complexity of component-machines for self-correcting networks. *Journal of Cybernetics*, 5:97–118, 1975.
200. L.Priese. On a simple combinatorial structure sufficient for sublying non-trivial self-reproduction. *Journal of Cybernetics*, 6:101–137, 1976.
201. L.Priese. On stable organization of normed networks. In *Proceedings of the third European meeting on cybernetics and systems research*, pages 381–394, 1976.
202. L.Priese. Reversible Automaten und einfache universelle 2-dimensionale Thuesysteme. *Zeitschr.f.math.Logik und Grundlagen der Math.*, 22:353–384, 1976.
203. L.Priese. Normed networks: Their mathematical theory and applicability. In *Applied General Systems Research*, NATO Conference Series vol.5, pages 381–394, 1978.
204. L.Priese. Towards a precise characterization of the complexity of universal and nonuniversal Turing machines. *SIAM J. Computing*, 8:508–523, 1979.
205. L.Priese. Modular implementation of concurrency. *International Journal of Theoretical Physics*, 21:993–1005, 1982.
206. L.Priese. On the concept of simulation in asynchronous, concurrent systems. *Progress in Cybernetics and Systems Research*, II, 1982. Proceedings of European Meeting on Cybernetics and Systems Research (Linz 1978).
207. L.Priese. Automata and concurrency. *Theor. Comp. Sci.*, 25:221–265, 1983.
208. L.Priese and D.Rödding. A combinatorial approach to self-correction. *J.Cybernetics*, 4:7–24, 1974.
209. M.Butler, A.Raschke, T.S.Hoang, and K.Reichel, editors. *Abstract State Machines, Alloy, B, TLA, VDM, and Z*, volume 10817 of *Lecture Notes in Computer Science*. Springer, Southampton (UK), 2018. 6th International Conference ABZ 2018.
210. M.Davis. *Computability and Unsolvability*. New York, 1958.
211. M.Davis, R.Sigal, and E.Weyuker. *Computability, Complexity, and Languages: Fundamentals of Theoretical Computer Science (2nd ed.)*. Elsevier Science and Technology, San Francisco (US), 1994. ISBN10 0122063821,ISBN13 9780122063824.
212. M.Deutsch. *Normalformen aufzählbarer Prädikate*. PhD thesis, Inst.math.Logik und Grundlagenforschung, Universität Münster, 1968.
213. M.Frappier, U.Glässer, S.Khurshid, R.Laleau, and S.Reeves, editors. *Abstract State Machines, Alloy, B, and Z - Second International Conference ABZ 2010*, volume 5977 of *Lecture Notes in Computer Science*, Orford,QC (Canada), 2010. Springer.
214. M. Minsky. Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines. *Annals of Mathematics*, 74:437–455, 1961.

215. M.Mölle and I.Schwank. Dimensional complexity and power spectral measures of the eeg during functional versus predicative problem solving. *Brain and Cognition*, 44:547 –563, 2000.
216. E. Moore. A simplified universal Turing machine. *ACM National Meeting (Toronto)*, pages 50–54, 1952. http://doi.acm.org/10.1145/800259.808993.
217. T. Neary, D. Woods, N.Murphy, and R.Glaschick. Wangs B machines are efficiently universal, as is Hasenjaegers small universal electromechanical toy. *J.of Complexity*, 30:634–646, 2014. https://doi.org/10.1016/j.jco.2014.02.003, ISSN 0885-064X.
218. N.Immerman. Descriptive and computational complexity. In J. Hartmanis, editor, *Computational Complexity Theory*, pages 75–91. American Math.Society, 1989.
219. NN. Heinrich Scholz. Biography. https://mathshistory.st-andrews.ac.uk/Biographies/Scholz/. Published at School of Mathematics and Statistics, University of St Andrews, Scotland.
220. NN. Mathematics Genealogy Project. https://genealogy.math.ndsu.nodak.edu/. Consulted July 14, 2021.
221. B. Pehrson and I. Simon, editors. *Technology and Foundations. Information Processing'94*, volume I, Track 4, Stream C: Evolving Algebras, Hamburg (Germany), 1994. Elsevier. Contains Proceedings of First ASM Workshop.
222. P.Koerber. *Untersuchung an sequentiellen, durch normierte Konstruktionen gewonnenen Netzwerken endlicher Automaten*. PhD thesis, Inst.math.Logik und Grundlagenforschung, Universität Münster, 1976.
223. P.Koerber and Th.Ottmann. Simulation endlicher Automaten durch Ketten aus einfachen Bausteinen. *EIK*, 10:133–148, 1974.
224. A. Raschke and D. Méry, editors. *Rigorous State-Based Methods*, volume 12709 of *Lecture Notes in Computer Science*, Ulm (Germany), 2021. Springer. 8th International Conference ABZ 2021.
225. A. Raschke, D. Méry, and F. Houdek, editors. *Rigorous State-Based Methods*, volume 1271 of *Lecture Notes in Computer Science*, Ulm (Germany), 2020. Springer. 7th International Conference ABZ 2020.
226. R.Fagin. *Contributions to the model theory of finite structures*. PhD thesis, University of California, Berkeley, 1973.
227. R.Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R.Karp, editor, *Complexity of Computation*, pages 43–73. SIAM-AMS Proceedings vol.7, 1974.
228. R.Moreno-Diaz and A.Quesada-Arencibia, editors. *Formal Methods and Tools for Computer Science. Eurocast 2001*, Las Palmas (Spain), 2001. IUCTC Universida de Las Palmas de Gran Canaria. Contains Extended Abstracts of 8th ASM Workshop. For selected full workshop papers see [45].
229. D. Rödding. Über die Eliminierbarkeit von Definitionsschemata in der Theorie der rekursiven Funktionen. *Zeitschr. Math. Logik Grundlagen der Mathematik*, 10:315–330, 1964.
230. D. Rödding. Einige äquivalente Präzisierungen des intuitiven Berechenbarbeitsbegriffs. *Math. Unterricht*, 11:21–38, 1965.
231. D. Rödding. Über Darstellungen der elementaren Funktionen II. *Arch. Math. Logik Grundlagenforsch.*, 9:36–48, 1966.
232. D. Rödding. Primitiv-rekursive Funktionen über einem Bereich endlicher Mengen. *Arch. Math. Logik Grundlagenforsch.*, 10:13–29, 1967.
233. D. Rödding. Registermaschinen. *Math.Unterricht*, 18:32–41, 1972.

234. D. Rödding. Modular decomposition of automata. In M.Karpinski, editor, *Proc. FCT-1983 Conference*, Lecture Notes in Computer Science vol.158, pages 394–412. Springer, 1983.
235. D. Rödding. Some logical problems connected with a modular decomposition theory of automata. In M. Richter, E. Börger, W. Oberschelp, B. Schinzel, and W. Thomas, editors, *Computation and proof theory*, Lecture Notes in Mathematics vol.1104, pages 365–388. Springer, 1984.
236. D. Rödding and E. Börger. The undecidability of $\forall\exists\forall(0,4)$ – formulae with binary disjunctions. *Journal of Symbolic Logic*, 39:412–413, 1974.
237. D. Rödding and H.Schwichtenberg. Bemerkungen zum Spektralproblem. *Zeitschr. f. math. Logik u. Grundlagen d. Math.*, 18:1–12, 1972.
238. D. Rödding and W. Rödding. Networks of finite automata. In *Proceedings of the third European meeting on cybernetics and systems research*, Progress in Cybernetics and Systems Research 1979. Hemisphere, Washington D.C., 1976.
239. W. Rödding. Netzwerke abstrakter Automaten als Modelle wirtschaftlicher und sozialer Systeme. *Schriftenreihe der Österreichischen Studiengesellschaft für Kybernetik*, 1975.
240. W. Rödding and H.Nachtkamp. On the aggregation of preferences to form a preference of a system. *Naval Research Logistic Quarterly*, 1978.
241. R.Vobl. Komplexitätsuntersuchungen an Basisdarstellungen endlicher Automaten. Diplomarbeit am Inst. für math. Logik und Grundlagenforschung in Münster, 1980.
242. R.W.Ritchie. Classes of predicatably computable functions. *Transactions American Math.Society*, 106:139–173, 1963.
243. K.-D. Schewe. Computation on structures: Behavioural theory, logic, complexity. In A.Raschke, E. Riccobene, and K.-D. Schewe, editors, *Logic, Computation and Rigorous Methods. Essays Dedicated to Egon Börger on the Occasion of His 75th Birthday*, volume 1275 of *Lecture Notes in Computer Science*, pages 266–282. Springer, 2021.
244. I. Schwank. Cognitive structures of algorithmic thinking. In *Proceedings of the 10th International Conference for the Psychology of Mathematics Education*, pages 404 – 409, 1986.
245. I. Schwank. $\alpha\beta\gamma$-automata realizing preferences. In E.Börger, editor, *Computation Theory and Logic*, volume 270 of *LNCS*, pages 320–333. Springer, 1987.
246. I. Schwank. Maschinenintelligenz: ein Ergebnis der Mathematisierung von Vorgängen Zur Idee und Geschichte der Dynamischen Labyrinthe. In C. Kaune, I. Schwank, and J. Sjuts, editors, *Mathematikdidaktik im Wissenschaftsgefüge: Zum Verstehen und Unterrichten mathematischen Denkens*, pages 30–72. Forschungsinstitut für Mathematikdidaktik in Osnabrück, 2005.
247. S.C.Kleene. General recursive functions of natural numbers. *Math.Ann.*, 112:727–742, 1936.
248. S.C.Kleene. *Introduction to Metamathematics*. North-Holland, 1952.
249. S.Cook. The complexity of theorem-proving procedures. In *Proc. 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.
250. J. Shepherdson and H. Sturgis. Computability of recursive functions. *J. ACM*, 10:217–255, 1963.
251. S.Stein and A.Wegener. Bericht über die Dienstreise nach Münster/W zur Durchsuchung des Korrespondenz-Nachlasses von Prof. Scholz. HNF - Heinz Nixdorf MuseumsForum, 2011.
252. R. F. Stärk, J. Schmid, and E. Börger. *Java and the Java Virtual Machine: Definition, Verification, Validation*. Springer-Verlag, 2001.

253. Th.Ottmann. Einfache universelle mehrdimensionale Turingmaschinen. Habilitationsschrift (Universität Karlsruhe).
254. Th.Ottmann. *Eine Theorie sequentieller Netzwerke*. PhD thesis, Inst.math.Logik und Grundlagenforschung, Universität Münster, 1971.
255. Th.Ottmann. über Möglichkeiten zur Simulation endlicher Automaten durch eine Art sequentieller Netzwerke aus einfachen Bausteinen. *Zeitschrift f.math.Logik und Grundlagen der Mathematik*, 19:223–238, 1973.
256. Th.Ottmann. Arithmetische Prädikate über einem Bereich endlicher Automaten. *Archiv f.math.Logik*, 16, 1974.
257. Th.Ottmann. Eine universelle Turingmaschine mit zweidimensionalem Band. *Elektronische Informationsverarbeitung und Kybernetik*, 11:27–38, 1975.
258. Th.Ottmann. Eine einfache universelle Menge endlicher Automaten. *Zeitschrift f.math.Logik und Grundlagen der Mathematik*, 24, 1978.
259. B. Trakhtenbrot. The impossibility of an algorithm for the decision problem for finite models. *Dokl. Akad. Nauk SSSR*, 70:596–572, 1950. English translation in: AMS Transl. Ser. 2, vol.23 (1963), p.1-6.
260. U.Glässer and P.Schmitt, editors. *Fifth International Workshop on Abstract State Machines*, Magdeburg (Germany), 1998. Otto-von-Guericke-Universität. Contains Proceedings of Fifth International ASM Workshop at Informatik'98.
261. U.Rohde. Computer für Anfänger.Teil 1. *mc*, 5:40–45, 1983.
262. B. van der Waerden. Denken ohne Sprache. In G.Révész, editor, *Thinking and Speaking*, pages 165–174. North-Holland, 1954.
263. W.Heinermann. *Untersuchungen über die Rekursionszahlen rekursiver Funktionen*. PhD thesis, Institut für math. Logik und Grundlagenforschung, Universität Münster, 1961.
264. C.-P. Wirth. A most interesting draft for Hilbert and Bernays Grundlagen der Mathematik that never found its way into any publication, and 2 cv of Gisbert Hasenjaeger. SEKI Working-Paper SWP201701 https://arxiv.org/pdf/1803.01386.pdf, 2017.
265. W.Oberschelp. Hans Hermes 12.2.1912 bis 10.11.2003. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 109:99–109, 2007.
266. D. Woods and T. Neary. The complexity of small universal Turing machines: A survey. *Theoretical Computer Science*, 410:443–450, 2009.
267. W.Rödding. Geschichte des Turingraums. Personal Communication to Egon Börger (November 8, 2021) and Letter of 27.4.2012 to Norbert Ryska from the Heinz Nixdorf MuseumsForum in Paderborn (Germany). Unpublished.
268. W.Schwabhäuser. *Entscheidbarkeit und Vollständigkeit der elementaren hyperbolischen Geometrie*. PhD thesis, Humboldt-Universität Berlin, 1960.
269. W.Schwabhäuser, W.Szmielew, and A.Tarski. *Metamathematische Methoden in der Geometrie*. Springer Verlag, 1983.
270. W.Zimmermann and B.Thalheim, editors. *Abstract State Machines 2004. Advances in Theory and Practice*, volume 3052 of *LNCS*. Springer, 2004. Contains Proceedings of 11th ASM Workshop (Lutherstadt Wittenberg).
271. Y.Ait-Ameur and K.-D.Schewe, editors. *Abstract State Machines, Alloy, B, TLA, VDM, and Z - 4th International Conference ABZ 2014*, volume 8477 of *Lecture Notes in Computer Science*, Toulouse (France), 2014. Springer.
272. Y.Gurevich, P.W.Kutter, M.Odersky, and L.Thiele, editors. *Abstract State Machines. Theory and Applications*, volume 1912 of *LNCS*, Monte Verità (Switzerland), 2000. Springer. Proceedings of 7th International ASM Workshop.

273. K. Zuse. Ansätze einer Theorie des allgemeinen Rechnens unter besonderer Berücksichtigung des Aussagenkalküls und dessen Anwendung auf Relaisschaltungen. https://digital.deutsches-museum.de/item/NL-207-0281/. Proposal for a doctoral dissertation submitted to H.Scholz. Unpublished.
274. K. Zuse. Mathematische Logik und Informatik. In *Proc. GI-5.Jahrestagung*, volume 34 of *Springer Lecture Notes in Computer Science*, pages 57–70, 1975.

# Forthcoming events

**Events Venue (unless otherwise specified)**:

> BCS, The Chartered Institute for IT
> Ground Floor, 25 Copthall Avenue, London, EC2R 7BP

The nearest tube station is Moorgate, but Bank and Liverpool Street are within walking distance as well.

Details of all forthcoming events can be found online here:

> https://www.bcs.org/membership/member-communities/facs-formal-aspects-of-computing-science-group/

Please revisit this site for updates as and when further events are confirmed.

# FACS Committee

Formal Aspects of Computing Science Specialist Group

**Jonathan Bowen**
FACS Chair and
BCS Liaison

**John Cooke**
FACS Treasurer and
Publications

**Roger Carsley**
Minutes Secretary

**Andrei Popescu**
LMS Liaison

**Ana Cavalcanti**
FME Liaison

**Brijesh Dongol**
Refinement
Workshop Liaison

**Keith Lines**
Government and
Standards Liaison

**Margaret West**
Inclusion Officer and
BCS Women Liaison

**Tim Denvir**
Co-Editor, FACS
FACTS

**Brian Monahan**
Co-Editor, FACS
FACTS

FACS is always interested to hear from its members and keen to recruit additional helpers. Presently we have vacancies for officers to help with fund raising, to liaise with other specialist groups such as the Requirements Engineering group and the European Association for Theoretical Computer Science (EATCS), and to maintain the FACS website. If you are able to help, please contact the FACS Chair, Professor Jonathan Bowen at the contact points below:

**BCS-FACS**
c/o Professor Jonathan Bowen (Chair)
London South Bank University
**Email:** jonathan.bowen@lsbu.ac.uk
**Web:** www.bcs-facs.org

You can also contact the other Committee members via this email address.

## Mailing Lists

As well as the official BCS-FACS Specialist Group mailing list run by the BCS for FACS members, there are also two wider mailing lists on the Formal Aspects of Computer Science run by JISCmail.

The main list <facs@jiscmail.ac.uk> can be used for relevant messages by any subscribers. An archive of messages is accessible under:

http://www.jiscmail.ac.uk/lists/facs.html

including facilities for subscribing and unsubscribing.

The additional <facs-event@jiscmail.ac.uk> list is specifically for announcement of relevant events.

Similarly, an archive of announcements is accessible under:

http://www.jiscmail.ac.uk/lists/facs-events.html

including facilities for subscribing and unsubscribing.

BCS-FACS announcements are normally sent to these lists as appropriate, as well as the official BCS-FACS mailing list, to which BCS members can subscribe by officially joining FACS after logging onto the BCS website.