Nieuwsbrief van de Nederlandse Vereniging voor Theoretische Informatica

Susanne van Dam* Joost-Pieter Katoen Joost Kok Jaco van de Pol
Femke van Raamsdonk

Inhoudsopgave

* CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands. Email: Susanne.van.Dam@cwi.nl

Van de Redactie

Geachte NVTI-leden!

Als u dit leest, bent u inmiddels begonnen aan de NVTI nieuwsbrief voor het jaar 2006. Wat kunt u zoal vinden in deze nieuwsbrief? Een belangrijk item is de aankondiging van de NVTI dag op 10 maart a.s. in Utrecht. Het programma is veelbelovend met befaamde buitenlandse sprekers zoals Kurt Mehlhorn en Martin Abadi (wie kent hen niet?) en de aansprekende sprekers van eigen bodem: Wan Fokkink en Mark de Berg. Kortom: genoeg redenen om 10 maart naar Utrecht te komen!

De bijdragen van IPA en SIKS geven een indruk over de activiteiten van deze onderzoekscholen in het afgelopen jaar. De technische bijdragen zijn van Freek Wiedijk (RUN) en Peter Flach (Univ. Of Bristol. UK). Hartelijk dank daarvoor.

Met onze dank aan CWI, NWO, Elsevier, IPA, SIKS en OzsL voor de sponsoring van de NVTI activiteiten, en Susanne van Dam en Anton Wijs voor de secretariële ondersteuning.

Namens de redactie en bestuur,

Joost-Pieter Katoen, hoofdredacteur
Joost Kok, voorzitter
Jaco van de Pol, secretaris

Samenstelling Bestuur

Prof. dr. Jos Baeten (TU/e)
Dr. Hans Bodlaender (UU)
Prof. dr. Harry Buhrman (CWI en UvA)
Prof. dr. ir. Joost-Pieter Katoen (RWTH en UT)
Prof. dr. Jan Willem Klop (CWI, RU en VU)
Prof. dr. Joost Kok (RUL) voorzitter
Prof. dr. John-Jules Meyer (UU)
Dr. Jaco van de Pol (CWI en TU/e) secretaris
Dr. Femke van Raamsdonk (VU)
Prof. dr. Grzegorz Rozenberg (RUL)
Prof. dr. Gerard Renardel de Lavalette (RUG)
Dr. Leen Torenvliet (UvA)

Invitation NVTI Theory Day, March 10, 2006

We are happy to invite you for the Theory Day 2006 of the NVTI. This event will take place on Friday March 10, 2006 from 9:30 until 16:45, at Hoog Brabant, Utrecht (close to Utrecht Central Station[1]).

As in previous years we have a strong program featuring excellent speakers from the Netherlands and abroad, covering important streams in theoretical computer science. Two lectures cover algorithms and complexity theory; the other two cover logic and semantics.

Lecturers:

Martín Abadi	Reconciling Two Views of Cryptography
Mark de Berg	I/O- and cache-efficient algorithms for spatial data
Wan Fokkink	Oh Mega Completeness
Kurt Mehlhorn	Reliable Geometric Computation

Please find below the program, the abstracts, a CV of the speakers and our sponsors.

Program

9.30-10.00: Arrival with Coffee
10.00-10.10: Opening
10.10-11.00: **Lecture:** Prof. Dr. Martin Abadi
(Microsoft Research and Un. of California at Santa Cruz)
Title: Reconciling Two Views of Cryptography
11.00-11.30: Coffee/Tea
11.30-12.20: **Lecture:** Prof. Dr. Mark T. de Berg (TU/e)
Title: I/O- and cache-efficient algorithms for spatial data
12.20-12.40: Dr. Mark Kas (NWO Exacte Wetenschappen/Physical Sciences)
The Research Agenda for Computer Science in The Netherlands
12.40-14.10: Lunch (see below for registration)
14.10-15.00: **Lecture:** Prof. Dr. Wan Fokkink (Free University and CWI)
Title: Oh Mega Completeness
15.00-15.20: Coffee/Tea
15.20-16.10: **Lecture:** Prof. Dr. Kurt Mehlhorn
(Max-Planck-Institute and University of Saarland)
Title: Reliable Geometric Computation
16.10-16.40: Business meeting NVTI

[1]From the big station hall, follow sign "Centrum". At "Radboudkwartier" it is only 100 meter. You find "Hoog Brabant" behind the info desk from shopping center "Hoog Catharijne".

Abstracts

Reconciling Two Views of Cryptography (Martín Abadi)

Two distinct, rigorous views of cryptography have developed over the years, in two mostly separate communities. One of the views relies on a simple but effective formal approach; the other, on a detailed computational model that considers issues of complexity and probability. There is an uncomfortable and interesting gap between these two approaches to cryptography. In this talk, we discuss this gap and the current research efforts that aim to bridge it. We focus on computational justifications for the formal treatment of encryption, and their recent applications to the study of guessing attacks and to XML access control.

I/O- and cache-efficient algorithms for spatial data (Mark de Berg)

Modern computer systems have a hierarchal memory consisting of a disk, main memory, and several levels of cache. The difference between the times to access these different levels of memory is quite large. This is holds in particular for main memory and disk: accessing the disk is typically about 100,000 times slower than accessing the main memory. Hence, it is important to take caching- and I/O-behavior into account when designing and analyzing algorithms. In this talk I discuss some of the recent results that have been obtained on I/O- and cache-efficient algorithms, focusing on algorithms and data structures for spatial data.

Oh Mega Completeness (Wan Fokkink)

In his CONCUR'90 paper, Rob van Glabbeek introduced sound and ground-complete axiomatizations for the basic process algebra BCCSP, modulo the semantics in his linear time - branching time spectrum. Also at CONCUR'90, Jan Friso Groote was the first to address whether these axiomatizations are omega-complete. Since then, positive and negative results have been obtained regarding the existence of omega-complete axiomatizations for BCCSP modulo the semantics in the aforementioned spectrum. In this talk I will give an overview of these results, the methods that were used to obtain them, and the remaining open questions.

Reliable Geometric Computation (Kurt Mehlhorn)

Reliable implementation of geometric algorithms is a notoriously difficult task. Algorithms are usually designed for the Real-RAM, capable of computing with real numbers in the sense of mathematics, and for non-degenerate inputs. But, real computers are not Real-RAMs and inputs are frequently degenerate. We start with a short collection of failures of geometric programs. We then go on to discuss approaches to reliable geometric computation:
– realization of an efficient Real-RAM as far as it is needed for geometry,
– design of algorithms with reduced arithmetic demand,
– controlled perturbation.

Curricula Vitae of the Lecturers

Martín Abadi is Professor of Computer Science at UCSC (since 2001) and Senior Researcher at Microsoft (since 2006). Earlier, he studied at Stanford University and worked at Digital's System Research Center and other industrial labs. His research is on computer and network security, programming languages, and specification and verification methods. He has contributed, for example, to the design and analysis of security protocols, to the foundations of object-oriented languages, and to temporal-logic verification techniques. His web page is at www.soe.ucsc.edu/ abadi/home.html.

Mark de Berg received an M.Sc. in computer science from Utrecht University in 1988, and he received a Ph.D. from the same university in 1992. Currently he is a full professor at TU Eindhoven. His main research interest is in computational geometry and its application to areas like GIS, computer graphics, and CAD/CAM. He is (co-)author of two books on computational geometry and he has published over 100 papers in journals and conferences.

Wan Fokkink obtained a PhD degree in Computer Science at the University of Amsterdam in 1994. After a postdoc at Utrecht University and a lectureship at the University of Wales Swansea, he became head of the Embedded Systems Group in CWI in 2000. Since 2004 he is full professor at the Free University in Amsterdam, and head of the Theoretical Computer Science Group. He is still affiliated to CWI one day a week.

Kurt Mehlhorn is Professor of Computer Science at the University of Saarland, and Director of the Max-Planck-Institute in Saarbrücken. His research interests are data structures, graph algorithms, computational geometry with exact computation, algorithm engineering, and theoretical complexity. He is the founder of many software libraries, among which the widely used LEDA software library.

Sponsors

The NVTI theory day 2006 is financially or otherwise sponsored by NWO (Netherlands Organisation for Scientific Research), Elseviers Science, CWI (Dutch Center of Mathematics and Computer Science) and the Dutch research schools IPA (Institute for Programming Research and Algorithmics), OzsL (Dutch Graduate school in Logic) and SIKS (Dutch research school for Information and Knowledge Systems).

Lunch registration

It is possible to participate in the organized lunch. Registration is required. Please register by E-mail (Susanne.van.Dam@cwi.nl) or by phone (020-5924189), no later than one week before the meeting (March 3). The costs of 15 Euro can be paid at the location. We just mention that in the direct vicinity of the meeting room there are plenty of nice lunch facilities as well.

www.win.tue.nl/ipa/

Institute for Programming research and Algorithmics

The research school IPA (Institute for Programming Research and Algorithmics) educates researchers in the field of programming research and algorithmics. This field encompasses the study and development of formalisms, methods and techniques to design, analyse, and construct software systems and components. IPA has three main research areas: Algorithmics & Complexity, Formal Methods, and Software Technology. Researchers from eight universities (University of Nijmegen, Leiden University, Technische Universiteit Eindhoven, University of Twente, Utrecht University, University of Groningen, Vrije Universiteit Amsterdam, and the University of Amsterdam), the CWI and Philips Research (Eindhoven) participate in IPA. In 1997, IPA was formally accredited by the Royal Dutch Academy of Sciences (KNAW). In 2002, this accreditation was extended for a period of five years.

On the European front, IPA cooperates with the research schools BRICS (Denmark), TUCS (Finland), UKII (United Kingdom), IP (Italy), GEFI (Germany), and FI (France) in the European Educational Forum (EEF).

Activities in 2005

IPA has two multi-day events per year, the Lentedagen and the Herfstdagen, which focus on a particular subject. In the 2002 - 2006 period, each of the Herfstdagen will be dedicated to one of IPA's four so-called application areas: Networked Embedded Systems, Security, Intelligent Algorithms, and Compositional Programming Methods. In 2005, the Herfstdagen were dedicated to Security, and the Lentedagen were on Software Architecture.

Lentedagen *March 30 - April 1, 2005, hotel De Korenbeurs, Made*
Software architecture is a topic that has figured in a low-key but consistent way throughout IPA events in the past years. After having looked at component-based architectures (Herfstdagen 1999), object-oriented architectures (Lentedagen on UML, 2000), and peer-to-peer systems (Lentedagen on Middelware, 2002), this year's Lentedagen had software architecture as the main topic, with the aim of presenting an overview of recent developments in the field that are of interest to the IPA community. The overall theme was architecture and change: dealing with change during design, the design of systems that change, and analysis of the architecture of existing systems (with the aim of changing them). The program contained sessions on variability, model driven and service oriented architectures, and the reconstruction and assessment of architectures. It was composed by Michel Chaudron (TU/e), Arie van Deursen (TUD,CWI), Arend Rensink (UT), and Eelco Visser (UU). Abstracts, hand-outs and papers are available through the Lentedagen webpage: `www.win.tue.nl/ipa/archive/springdays2005/`.

Herfstdagen on Security *November 21-25,Hotel Zwartewater,Zwartsluis*
IPA first organised an event on Security in 2001 (Lentedagen). The problems with computer

security that made this topic highly relevant to society then, persist today. However, the insight that Security is not an add-on feature, but a fundamental aspect of software engineering, has gained ground. Based on the experience of the Lentedagen, Security was chosen as one of IPA's four main application areas for the 2002-2006 period, and an initial general description of the area was produced (see `www.win.tue.nl/ipa/applicationareas`). Since then, the area has proven to be fertile ground for the IPA research community and Security research has grown considerably both within and around IPA.

This year's Herfstdagen looked at the current state of the field. The form of the program reflects the fact that Security is about striking a balance between different, often conflicting, concerns. Two such pairs of concerns, Identification & Privacy and Vulnerabilities & Resilience, are the theme for the main parts of the program. Each part contained sessions on technology, theory (concepts and formalisms) and applications. The sessions integrated contributions on academic and industrial research with talks on societal aspects. Their themes were: biometrics, privacy & anonimity, identification of civilians, RFID, electronic voting, software vulnerabilities, trust management, secure data management, protocols, and security in organisations. The program was composed by Pieter Hartel (UT), Bart Jacobs (RU), and Sjouke Mauw (TU/e). Abstracts, hand-outs, and papers are available through the website at `www.win.tue.nl/ipa/archive/falldays2005/`.

IPA organises Basic Courses on each of its major research fields, Algorithms and Complexity, Formal Methods and Software Technology. These Basic Courses intend to give an overview of (part of) the research of IPA in these fields. In 2005, the first course of a new cycle was took place in Eindhoven.

Basic Course on Algorithms and Complexity *January 31 - February 4, TU/e*
The Basic Course focussed on five subjects areas in algorithmics where succesfull research is being conducted by groups in IPA: operations research, graph and network algorithms, natural computation, and alternative computational models. To each topic an entire course day was dedicated. The final course day addressed an interesting application area for algorithmic techniques. Topics and teachers were: *machine scheduling* Han Hoogeveen (UU), *randomized geometric algorithms* Mark de Berg (TU/e), *evolutionary algorithms* Han La Poutr and Peter Bosman (CWI), *quantum computing* Harry Buhrmann (CWI), *bio-informatics* Hendrik Jan Hoogeboom and Walter Kosters (UL). More information on the course program is available through the webpage: `www.win.tue.nl/ipa/archive/algbasiccourse2005/`

In addition to organising its own activities, IPA sponsored 4 further events: the Theoriedag 2005 of the NVTI (March 4, Utrecht), the first meeting of the Security PhD Association Netherlands (May 27, TUD), the doctoral symposium of the fifth International Conference on Integrated Formal Methods (IFM2005, November 29-December 2, TU/e), and the fourth international symposium on Formal Methods for Components and Objects (FMCO 2005, November 1-4, CWI).

IPA Ph.D. Defenses in 2005

Erika Ábrahám (UL, 20 January), *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Promotores: prof.dr. J.N. Kok, prof.dr. W.-P. de Roever. Co-promotores: dr. F.S. de Boer, dr. M. Steffen. IPA-Dissertation Series 2005-01.

Cheun Ngen Chong (UT, 4 February), *Experiments in Rights Control - Expression and Enforcement.* Promotor: prof.dr. P.H. Hartel. Co-promotor: dr. S. Etalle. IPA-Dissertation Series 2005-03.

Ronald Ruimerman (TU/e, 14 February), *Modeling and Remodeling in Bone Tissue.* Promotores: prof.dr. P.A.J. Hilbers, prof.dr.ir. R. Huisken. IPA-Dissertation Series 2005-02.

Hui Gao (RuG, 15 April), *Design and Verification of Lock-free Parallel Algorithms.* Promotores: prof.dr. W.H. Hesselink, prof.dr.ir. J.F. Groote. IPA-Dissertation Series 2005-04.

Ivan Kurtev (UT, 19 May), *Adaptability of Model Transformations*. Promotor: prof.dr. M. Aksit. IPA-Dissertation Series 2005-08.

Mugurel Ionita (TU/e, 31 May), *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Promotores: prof.dr.dipl.-ing. D.K. Hammer, prof.dr.ir. J.F. Groote. Copromotor: dr.ir. P.H.M. America. IPA-Dissertation Series 2005-06.

Harm van Beek (TU/e, 9 June), *Specification and Analysis of Internet Applications*. Promotor: prof.dr. J.C.M. Baeten. Co-promotor: dr. S. Mauw. IPA-Dissertation Series 2005-05.

Thomas Wolle (UU, 13 June), *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Promotor: prof.dr. J. van Leeuwen. Co-promotor: dr. H. Bodlaender.. IPA-Dissertation Series 2005-09.

Olga Tveretina (TU/e, 29 June), *Decision Procedures for Equality Logic with Uninterpreted Functions*. Promotor: prof.dr.ir. J.F. Groote. Co-promotor: dr. H. Zantema. IPA-Dissertation Series 2005-10.

Gabriele Lenzini (UT, 30 June), *Integration of Analysis Techniques in Security and Fault-Tolerance*. Promotor: prof.dr. P. Hartel. Co-promotor: dr. S. Etalle. IPA-Dissertation Series 2005-07.

Anthony Liekens (TU/e, 5 July), *Evolution of Finite Populations in Dynamic Environments*. Promotores: Prof.dr. P.A.J. Hilbers, Prof.dr. E.H.L. Aarts. Co-promotor: Dr.ir. H.M.M. ten Eikelder. IPA-Dissertation Series 2005-11.

Jeroen Eggermont (UL, 14 September), *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Promotor: Prof.dr. J.N. Kok. Co-promotor: Dr. W.A. Kosters. IPA-Dissertation Series 2005-12.

Bastiaan Heeren (UU, 20 September), *Top Quality Type Error Messages*. Promotor: Prof.dr. S. Swierstra. Co-promotor: Dr. J. Hage. IPA-Dissertation Series 2005-13.

MohammadReza Mousavi (26 September), *Structuring Structural Operational Semantics*. Promotores: Prof.dr.ir. J.F. Groote, prof.dr. G.D. Plotkin. Co-promotor: dr.ir.M.A. Reniers. IPA-Dissertation Series 2005-15.

Goran Frehse (RU, 10 October), *Compositional Verification of Hybrid Systems using Simulation Relations*. Promotores: Prof.dr. F.W. Vaandrager, prof.dr. S. Engel. IPA-Dissertation Series 2005-14.

Ana Sokolova (TU/e, 3 November), *Coalgebraic Analysis of Probabilistic Systems*. Promotor: Prof.dr. J.C.M. Baeten. Co-promotor: Dr. E.P. de Vink. IPA-Dissertation Series 2005-16.

Tjalling Gelsema (UL, 8 November, *Effective Models for the Structure of pi-Calculus Processes with Replication*. Promotor: Prof.dr. G. Rozenberg. Co-promotor: Dr. J. Engelfriet. IPA-Dissertation Series 2005-17.

Atze Dijkstra (UU, 14 November), *Stepping through Haskell*. Promotor: Prof.dr. S.D. Swierstra. IPA-Dissertation Series 2005-21.

Jurgen Vinju (UvA, 15 November), *Analysis and Transformation of Source Code by Parsing and Rewriting*. Promotor: Prof.dr. P. Klint. Co-promotor: Dr. M.G.J. van den Brandt. IPA-Dissertation Series 2005-19.

Peter Zoeteweij (UvA, 29 November), *Composing Constraint Solvers*. Promotores: Prof.dr. K.R. Apt, prof.dr. F. Arbab. IPA-Dissertation Series 2005-18.

Yee Wei Law (UT, 1 December), *Key Management and Link-layer Security of Wireless Sensor Networks: Energy-efficient Attack and Defense*. Promotor: prof.dr. P. Hartel. Co-promotor: dr. S. Etalle. IPA-Dissertation Series 2005-22.

Miguel Valero Espada (VUA, 5 December), *Modal Abstraction and Replication of Processes with Data*. Promotor: Prof.dr. W. Fokkink. Co-promotor: Dr. J.C. van de Pol. IPA-Dissertation Series 2005-20.

Activities in 2006

The themes for the first major IPA-events of 2006 is known: the Lentedagen will be on Testing. In addition, IPA is planning to stage two of its Basic Courses this year (Formal Methods and Software Technology). The first Basic Course and the Lentedagen have a definite schedule. More information on all upcoming activities will become available through the IPA web site.

Basic Course on Formal Methods *January 16 - 20, TU/e.*
The Basic Cours focusses on five main areas of Formal Methods research. One course day is dedicated to each of these areas, in which lectures are combined with hands-on tool training. Topics, tools and teachers are: *introduction* Jos Baeten (TU/e); *interface specification (ISPEC) & Calisto* Hans Jonkers (Philips Research), Ruurd Kuiper & Erik Luit (TU/e); *model checking & SPIN*, Joost-Pieter Katoen (RWTH Aachen), *theorem proving & PVS*, Erik Poll (RU), *process algebra & mCRL2*, Jan Friso Groote (TU/e), *business processes & Petri nets*, Wil van der Aalst (TU/e). More information on the course program is available through the website: `www.win.tue.nl/ipa/activities/fmbasiccourse2006/`.

Lentedagen *April 19-21.*
The aim of the Lentedagen is to present an overview of the current research on testing in and around IPA, with an emphasis on model-driven testing. The program is composed by Jaco van de Pol (CWI), Judi Romijn (TU/e), Mariëlle Stoelinga (UT), and Jan Tretmans (RU). More information will become available through the Lentedagen webpage as the program develops, `www.win.tue.nl/ipa/activities/springdays2006/`.

Addresses

Visiting address
Technische Universiteit Eindhoven
Main Building HG 7.22
Den Dolech 2
5612 AZ Eindhoven
The Netherlands

Postal address
IPA, Fac. of Math. and Comp. Sci.
Technische Universiteit Eindhoven
P.O. Box 513
5600 MB Eindhoven
The Netherlands

tel. (+31)-40-2474124 (IPA Secretariat)
fax (+31)-40-2475361
e-mail `ipa@tue.nl`
url `www.win.tue.nl/ipa/`

School for information and Knowledge Systems in 2005

Richard Starmans (UU)

Introduction

SIKS is the Dutch Research School for Information and Knowledge systems. It was founded in 1996 by researchers in the field of Artificial Intelligence, Databases & Information Systems and Software Engineering. Its main concern is research and education in the field of information and computing sciences, more particularly in the area of information and knowledge systems (IKS). SIKS is an interuniversity research school that comprises 12 research groups from 10 universities and CWI. When SIKS received its accreditation by KNAW in 1998, only 35 Ph.D. students and about 70 research fellows were involved in the school. Currently, nearly 350 researchers are active, including 170 Ph.D.-students. The Vrije Universiteit in Amsterdam is SIKS' administrative university. The office of SIKS is located at Utrecht University In June 2003 SIKS was re-accreditated by KNAW for another period of 6 years.

Organisation:

In 2005 Prof .dr. R.J. (Roel) Wieringa (University of Twente) has been appointed as the new scientific director of SIKS. As of January 1 2006 he stepped into the shoes of prof. dr. J.-J. (John-Jules) Ch. Meyer (Utrecht University), who held this position for nearly a decade.
Roel Wieringa studied mathematics and philosophy and obtained his phd at the Vrije Universiteit in Amsterdam in 1990. His dissertation was entitled “Algebraic Foundations for Dynamic Conceptual Models.” In 1998 he became a full professor at Twente University, where he currently holds the Chair “Ïnformation Systems” at the Faculty of Electrical Engineering, Mathematics and Computer Science. He has been a member of the board of governors of SIKS as of 1998. Wieringa has accepted the appointment for a period of three years.
John-Jules Meyer will keep his position at the scientific board of SIKS as focus director for Agent systems, one of SIKS' eight research foci.

Activities

We here list the main activities (co-)organized or co-financed by SIKS. We distinguish basic courses, advanced courses and other acitivities (including masterclasses, workshops, one-day seminars, conferences and research colloquia)

Basic courses

“Research methods and methodology for IKS", November, 21-23, 2005 Woudschoten, Zeist
Course directors: dr. H. Weigand (UvT), prof. dr. R. Wieringa (UT),
prof. dr. J-J.Ch. Meyer, dr. R. Starmans (UU)

“Combinatory Methods”, May 09-11, 2005, Landgoed Huize Bergen, Vught
Course director: dr. N. Roos (UM)

“Learning and Reasoning”, May 11-13, 2005, Landgoed Huize Bergen, Vught
Course director:dr. A. Ten Teije (VU)

“Formal methods for IKS”. December 19-21, 2005, Landgoed Huize Bergen, Vught
Course directors: prof. dr. E.O. Postma (UM), prof. dr. J.-J, Ch. Meyer (UU)

“Agent systems” December 21-23, 2005, Landgoed Huize Bergen, Vught
Course directors: prof.dr. C. Jonker (RUN), prof. dr. J.-J, Ch. Meyer (UU)

Advanced courses

“Computational Intelligence”, February 17-18 2005, Woudschoten, Zeist
Course directors: prof.dr. A.P.J.M. Siebes (UU), dr. U. Kaymak (EUR)

"XML: where databases and information retrieval meet" , April 18-19, 2005, Leusden
Course directors: Dr. ir. D. Hiemstra (UT), Dr. ir. M. van Keulen (UT)

“Summer course on Datamining”, June 27- July 01, 2005, Maastricht
Course directors: Dr. E. Smirnov (UM), Dr. J. Donkers (UM), Prof. dr. E.O. Postma (UM)

“Business Process Integration", September 19-20, 2005 Best Western Dish hotel, Enschede
Course director: dr. H. Weigand (UvT)

Other activities

- 10-11 January 2005 Workshop on Information Retrieval, DIR 05, Utrecht
- 26 January 2005 SIKS-IKAT Symposium "Go at the frontiers of AI", Maastricht
- 17-18 February 2005 Benelearn 2005, Enschede
- 15 March 2005 Symposium: Waarheid in Taal, Amsterdam
- 19 May 2005 IKAT-SIKS Symposium: Machine Learning for Commercial Game AI , Maastricht
- 18-22 Juli 2005 EASSS 2005; agent systems summer school, Utrecht
- 17-18 October 2005 BNAIC 2005, Brussel
- 31 October 2005 Dutch-Belgian Database Day, Amsterdam
- 11 November 2005 SIKS-day 2005, Utrecht
- 23 November Symposium on Computer science and Law, Leiden
- 30 November 2005 IKAT-SIKS Symposium: The nature of representation, Maastricht
- 20 December 2005 Symposium on E-commerce and fair trade principles, Leiden
- SIKS-IKAT Research colloquium: organized 6 times in Maastricht
- CABS-SIKS Research colloquium, organized 6 times in Delft en Utrecht
- IKS-SIKS Information science seminar, organized 7 times in Utrecht

Promotions in 2005

In 2005 21 researchers successfully defended their Ph.D.-thesis and published their work in the SIKS-dissertation Series.

2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications
Promotor: Prof. dr. B.J. Wielinga (UVA)
Co-promotor: dr. M.W. van Someren (UVA)
Promotion: 28 January 2005

2005-02 Erik van der Werf (UM)
AI techniques for the game of Go
Promotor: prof. dr. H.J. van den Herik (UM)
Co-promotor: dr. J.W.H.M. Uiterwijk (UM)
Promotion: 27 January 2005

2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language
Promotores: prof. dr. ir. Th. P. van der Weide (RUN), prof. C.H.A. Koster (RUN)

Promotion: 26 January 2005

2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data
Promotores: prof. dr. P.M.G. Apers (UT)
Co-promotor: Dr. Ir. R. A. de By (ITC)
Promotion: 23 February 2005

2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing
Promotores: prof. dr. M. de Rijke (UVA), prof. dr. R. Scha (UVA)
Promotion: 06 April 2005

2005-06 Pieter Spronck (UM)
Adaptive Game AI
Promotores: prof.dr. H.J. van den Herik (UM), prof.dr. E.O. Postma (UM)
Promotion: 20 May 2005

2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems
Promotores: Prof.dr. P. De Bra (TUE) Prof.dr.ir. G-J. Houben (VUB/TUE)
Co-promotor: Prof.dr. J. Paredaens (TUE/UA)
Promotion: 20 June 2005

2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications
Promotores: Prof.dr. P. De Bra (TUE) Prof.dr.ir. G-J. Houben (VUB/TUE)
Co-promotor: Prof.dr. J. Paredaens (TUE/UA)
Promotion: 20 June 2005

2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages
Promotor: Prof.dr. F. van Harmelen (VU)
Promotion: 04 July 2005

2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments
Promotores: Prof. dr. B. J. Wielinga (UVA), Prof. dr. J. A. P. J. Breuker (UVA)
Co-promotor: Dr. B. Bredeweg (UvA)
Promotion: 06 July 2005

2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search
Promotores: prof.dr. F.M.T. Brazier (VU), prof.dr.ir. M.R. van Steen (VU)
Promotion: 05 April 2005

2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry
Promotor: prof.dr. A. de Bruin (EUR)
Prof.dr.ir. A. Verbraeck (Delft University/University of Maryland)
Promotion: 21 Oktober 2005

2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen

Promotores: prof.dr. H.J.van den Herik (UM/UL), prof.dr.H.M.Dupuis (UL), prof.Dr.E.O.Postma (UM)
Promotion: 24 November 2005

2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics
Promotores: prof dr. A.Th. Schreiber (VU), prof dr. J.M. Akkermans (VU)
Promotion: 12 October 2005

2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes
Promotores: Prof. dr. J. Treur (VU) , Prof. dr. C.M. Jonker (RUN)
Promotion: 23 November 2005

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages
Promotor: Prof.dr.ir. G.J. van der Steen (UU)
Co-promotor: dr. H. van Oostendorp (UU)
Promotion:17 October 2005

2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components
Promotor: prof.dr.ir. J.L.G. Dietz (TUD)
Promotion: 26 September 2005

2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks
Promotor: Prof.dr.ir L. C. van der Gaag (UU)
Promotion:17 October 2005

2005-19 Michel van Dartel (UM)
Situated Representation
Promotores:prof.dr. E.O. Postma (UM), prof.dr. H.J. van den Herik (UM)
Promotion: 1 December 2005

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives
Promotores:prof.dr. H.J. van den Herik (UM), prof.dr. G. Howells (Sheffield)
Promotion: 20 December 2005

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics
Promotor: Prof.dr. W. Jonker (UT)
Promotion:16 November 2005

On the usefulness of formal methods

Freek Wiedijk

Institute for Computing and Information Sciences
Radboud University Nijmegen

The editor of this newsletter has asked me to write something for it. Now it would be easiest for me to just write about my research subject, *formalization of mathematics*, and how that field will fundamentally change the way that mathematicians look at proof. ('There will be a time when mathematicians will only consider something to be "proved" if the proof has been encoded in full detail in a computer, all the way to the axioms of set theory, and if that encoding has been completely checked by that computer. When that time has arrived, a mathematical article will only be accepted for publication if it is accompanied by a formal, computer-checked counterpart.[1] The referees then only will have to judge whether the result is new and interesting, and not whether it is correct ... because that will then be known already.')

However, that will not be the main topic of this essay. Instead, I will focus on the subject of the usefulness of working on technology for proving *software* correct. That is, the usefulness of using methods from mathematical logic to develop a technology for creating programs that are free of 'bugs'.[2]

Some years ago I was working on the formalization of the Fundamental Theorem of Algebra[3] in the Coq system. A *formalization* is an encoding of a mathematical proof in such detail, that the computer can verify the mathematical correctness without any further human interference. When one formalizes a proof, it is not the computer that provides the proof. Instead it is the human who 'guides' the

[1] In the field of mathematical logic this already occasionally happens today.

[2] In the NOAG-ict 2005–2010, which is the research agenda for Dutch computer science research, the field of formal methods does not have a research theme of its own. It has been put there as part of theme number six, *intelligente systemen*, the theme for artificial intelligence. To me that shows a fundamental misunderstanding of where the main promise of formal methods lies, what formal methods is all about. It really should have been in the theme that contains the study of algorithms and programming languages, theme number seven: *methoden voor ontwerpen en bouwen*.

[3] The Fundamental Theorem of Algebra was the subject of the 1799 PhD thesis of Carl Friedrich Gauss. It states that the field of the complex numbers is 'algebraically closed', which is a compact way of saying that every non-constant polynomial has a zero, and that therefore every polynomial can be written as a product of linear factors. The Fundamental Theorem of Algebra has three formalizations, which were all finished in the year 2000. The first formalization was done using the Polish system Mizar by Robert Milewski. The second formalization was done using the British system HOL Light by John Harrison. The third formalization was done using the French system Coq by Herman Geuvers, with a group of people to which I also belonged. This third formalization encoded a proof that is more complicated than the usual ones, because it also is *intuitionistically* valid.

proof, although he is supported by the automation of routine proof tasks that the computer provides.

Writing a formalization is an activity that is very similar to writing a large computer program. The languages that one uses when formalizing are very similar to programming languages, and the activity of modelling a proof in such a language is also very similar to the activity of programming. To show the kind of code that one writes when formalizing, here are the final lines in the Coq proof of the Fundamental Theorem of Algebra:

```
...
intro H0. apply H0.
intro i. generalize (Hs i).
intro H1; inversion_clear H1; assumption.
exact (seq_is_CC_Cauchy n H0n q qnonneg qlt10 (AbsCC p ! Zero[+]One)
  Hp s Hs2).
intro i; generalize (Hs i); intro Ha; elim Ha; intros; assumption.
exact (less_plusOne _ (AbsCC p ! Zero)).
apply zero_lt_posplus1.
apply AbsCC_nonneg.
Qed.
```

Note that this looks much like 'computer code'. However, there is a *big* difference between writing a computer program and writing a formalization. When one writes a formalization, one can be completely sure that it will be correct. In contrast with this a program of a non-trivial size will always have 'bugs'. (Every programmer knows that.[4])

At a recent Dagstuhl conference about the use of computers in mathematics, a mathematician in the audience who did not know much about formalization could not believe that this kind of total correctness is possible. He clearly believed that because formalization is a human activity, it surely should be taken into account that there would be mistakes. But no. (For instance, I am quite confident that our formalization of the Fundamental Theorem of Algebra is 100% correct.) At that same conference there was a girl who had formalized something, also using the Coq system. *She* knew about formalization. And she confidently proclaimed that she would be willing to bet 100,000 dollars that her work was totally free of errors. Now I would not go *that* far that I would bet such an amount on it, but I still am sure that she was right about the correctness of her work.[5]

[4] When I learned how to program, I was very much surprised that one makes mistakes all the time. My programs would often behave in ways that to me seemed completely impossible, even when I took into account that there would be some mistakes in it. (Of course like everyone I got used to this phenomenon, but initially this was very surprising to me.)

[5] Let me analyze what are the possibilities for mistakes when doing a formalization. I think that there are three. The first is the possibility of bugs in the software that checks the formalization. Now of course there will be bugs in that software. But it is rather unlikely that such bugs would *accidentally* allow someone to have

When we were formalizing the Fundamental Theorem of Algebra, we were using the versioning system CVS to keep track of our files. With this system a group of people can all work on a bunch of files simultaneously. When I described CVS to a programmer friend who had no experience with it, he told me that with such a system he would very much worry about different people making inconsistent changes to the files. And then I told him that if I would use CVS for programming, then I would worry about that too. But we used CVS with Coq, and therefore we did not have to worry about this at all! We used the rule that one was only allowed to check in files when the whole set was accepted by the system as being correct. This meant that at any time the files in the system would be a consistent whole. And therefore one could work with CVS without having to worry about modifications of someone else in the group 'breaking' something that one had done.

The experience of formalizing with Coq was wonderful. It was like programming, but then knowing that there were no mistakes, that there were no bugs at all.

But then, after the formalization of the Fundamental Theorem of Algebra was finished, I went back to 'normal' programming. That was scary! It was like someone had removed the safety net: I could make mistakes again!

Now the moral of this story is that I think that in working with Coq I have tasted the future of programming. Eventually (and I do not say that I know how or when) I expect the technology of programming to develop into a form that gives one the experience that I had with Coq. When that happens, programs will generally be free of bugs. Today this sounds unthinkable, but I really do believe that this will come about.

---

the system proclaim incorrect mathematics to be correct. Problems with normal programs are hardly ever caused by compiler bugs either. Also, the systems that check formalizations for correctness generally have an architecture that localizes the correctness of the check to a very small part of the program, a 'proof checking micro-kernel'. (Henk Barendregt calls this principle of having such a micro-kernel the *de Bruijn criterion*, after the pioneer of formalization N.G. de Bruijn.) This makes it even more unlikely that bugs in the checker will allow incorrect mathematics to accidentally 'slip through' the system.

The second possibility to have incorrect mathematics accidentally be accepted by a proof checking system is that the logical foundations of that system are inconsistent. While this certainly is not unthinkable (for instance, the Coq system is closely related to the logical systems of Per Martin-Löf, whose very first logical system happened to be inconsistent; so maybe his current crop of logics also is inconsistent, and we just are not smart enough to see this), again I think it is very unlikely that because of this one would prove falsehoods *by accident*.

The third possibility to make 'mistakes' in a formalization is that the definitions of the notions that one reasons about are not what one thinks them to be. This is the only significant problem of the three. However, once one proves many lemmas about a small number of definitions, one can be quite confident that these definitions mean what the formalizer thinks they mean.

I expect this development to come from the culture of the research field that is called 'formal methods'. Eventually. It is the reason that I think formal methods are a useful part of computer science.

Does this mean that I am advocating people to download Coq and start using it to prove their programs correct. No, certainly not! What I am saying is that the culture of proof assistants like Coq will be a fertile ground from which a new kind of programming will develop eventually. I am *not* saying that these proof assistants already are useful for more than proving the correctness of rather simple programs (programs like the `Vector` class from the Java library; proving that correct does not seem very important for practical work.)

Still, researchers that use proof assistants are already doing interesting things with it. For example there is the 'Cminor' compiler from INRIA, which has been built by Xavier Leroy. This compiler, called Compcert, has been proved correct using Coq (so there is a real possibility that its correctness approaches the correctness of our Fundamental Theorem of Algebra formalization). It compiles a realistic subset of the C programming language called Cminor to PowerPC assembly code, applying non-trivial code optimizations on the way. Xavier Leroy is the man behind one of the best compilers of functional languages in the world – the ocaml compiler of the ML language – and I consider it very interesting that he wants to spend his time on using proof checking technology for creating a provably correct program.

Another program, one that can *really* be relied on to be correct is the Four Color Theorem[6] program by Georges Gonthier. (At some point Georges Gonthier went to work for Microsoft. Please note: the most impressive formalization in the world is owned by Microsoft!) The Four Color Theorem was proved in the seventies using a very controversial approach. Part of the proof was the running of a computer program that went through millions of graph colorings. This program ran for days, then, and a modern version of that program, coded in C, still takes minutes to run. At that time the mathematicians were not satisfied by this approach. They wondered how one could be sure that the program did not contain bugs. What Georges Gonthier did was rewrite the program in a purely functional subset of ML, and then prove that version correct in Coq. And he did more: he also formalized all the graph theory and topology needed to prove a very clean and natural version of the Four Color Theorem statement. (This proof was an extension of the correctness proof of his version of the program.) In case you are interested, the statement that he proved was:

```
forall m : map R, simple_map m -> map_colorable 4 m
```

This was formalized in a way that the micro-kernel of the Coq system can check the correctness of the formalization all the way down to the Coq axioms. And to do this checking Coq also runs the – provably correct – program that is part of

[6] The Four Color Theorem states that every 'map' can be colored using four colors in such a way that two neighboring countries always have a different color. For a very long time this was a famous open problem, until it finally was proved in 1975 by Kenneth Appel and Wolfgang Haken.

the formalization. The running of that program only takes on the order of days (on an ordinary PC.)

In case you wonder how the notions `map`, `simple_map` and `map_colorable` that occur in this statement are defined, here are some relevant definitions that I lifted from Gonthier's files:

```
Inductive point : Type := Point (x y : R).

Definition region : Type := point -> Prop.

Definition intersect (r1 r2 : region) : region :=
  fun z => r1 z /\ r2 z.

Definition subregion (r1 r2 : region) := forall z, r1 z -> r2 z.

Definition map : Type := point -> region.

Definition inmap (m : map) : region := fun z => m z z.

Definition covers (m m' : map) := forall z, subregion (m z) (m' z).

Definition size_at_most n m :=
  exists f, forall z, inmap m z -> exists2 i, i < n & m (f i) z.

Record proper_map (m : map) : Prop := ProperMap {
  map_sym : forall z1 z2, m z1 z2 -> m z2 z1;
  map_trans : forall z1 z2, m z1 z2 -> subregion (m z2) (m z1)
}.

Record simple_map (m : map) : Prop := SimpleMap {
  simple_map_proper :> proper_map m;
  map_open : forall z, open (m z);
  map_connected : forall z, connected (m z)
}.

Definition border (m : map) z1 z2 :=
  intersect (closure (m z1)) (closure (m z2)).

Definition corner_map m z : map :=
  fun z1 z2 => m z1 z2 /\ closure (m z1) z.

Definition not_corner m z := size_at_most 2 (corner_map m z).

Definition adjacent m z1 z2 := meet (not_corner m) (border m z1 z2).

Record coloring (m k : map) : Prop := Coloring {
  coloring_proper :> proper_map k;
  coloring_inmap : subregion (inmap k) (inmap m);
  coloring_covers : covers m k;
  coloring_adj : forall z1 z2, k z1 z2 -> adjacent m z1 z2 -> m z1 z2
}.

Definition map_colorable n m :=
  exists2 k, coloring m k & size_at_most n k.
```

(I left out the definitions of topological notions like `open`, `connected` and `closure`, to prevent this list of definitions from becoming too long. These notions mean what you would expect them to mean.)

Now the Four Color Theorem program is certainly not a trivial program. The ML version that Georges Gonthier proved correct is around 2,500 lines long (even the C version is around 1,000 lines), and it uses very smart tricks to make it run as fast as possible. It certainly has not been designed to be straight-forward, to make the proofs easy. That it can be proved correct shows that proof checking of programs has reached some maturity.

Various technologies have been developed for proving programs correct. They fall roughly in two classes:

- Either one takes existing programming technology, and then *adds* a layer on top of that, which guarantees that the program is correct. This means that one writes a program like one always does it, but then uses software that analyzes it in some way (often the *statement* that is proved about the program then is just something like 'all array indices will stay within bounds and no nil pointers will be dereferenced'[7]), or maybe annotates it with invariants and then generates a series of lemmas from that which then can be proved using a proof assistant.[8]
- Alternatively, one can change the way one develops software, to make it more abstract, more *mathematical*. For instance one might restrict programming to a purely functional language (as is the case both with the Cminor compiler and with the program from Georges Gonthier's Four Color Theorem formalization). Or one might require the user to develop his or her program by first making an abstract specification, and then *refine* that specification to the real program. (As far as I understand it, that is what one does when using the 'B method'[9]).

[7] An example of a system like this is Microsoft's SDV (Static Driver Verifier), see

`http://research.microsoft.com/slam/`

Apparently at Microsoft some people seem to be thinking that formal methods are useful for analyzing production code. Bill Gates even is quoted calling the use of 'actual proof about the software' to be 'the Holy Grail of computer science'.

[8] A very nice example of systems that implement this are Jean-Christophe Filliâtre's Why and Caduceus tools. See the web page at:

`http://why.lri.fr/caduceus/index.en.html`

It is surprising to me that there is not more work being done on systems that follow this general approach.

[9] The B method was used to prove the software correct that manages a line of the Paris metro that has driverless trains. (One would like not to have metro trains collide because of bugs in the software.) This is most that I know about the B method, really. Well, I also know that the logical foundations that it uses is a variant of ZF set theory.

> Or, a method that used to be popular in the type theoretical community (although no one believes much in it anymore): one can just take a formalization of a proof, and then automatically *extract* a program from that.[10]

I do not know very well what position I take between these two approaches. On the one hand I do not think these methods will find wide adoption when programmers are forced into a mathematical straight-jacket, when they are required to program with their hands bound behind their backs so to say. For instance, I *certainly* do not think that *purely* functional programming (where you have to model all input/output, mutable data structures, exceptions etc. using so-called *monads*) will ever conquer the world.

On the other hand I think that the most efficient way to really reduce the number of software problems is *not* to have programmers prove things about programs the way they are now, but instead to have them change to a bit more abstract world view. Let me try to give two examples of what I mean by this.

I would expect that many of the problems with the – surprisingly popular – Microsoft Windows platform (bugs; but also the proliferation of viruses and worms that occurs there) often are caused by simple things like buffer overruns and dereferencing of nil pointers. Now suppose that Windows was not programmed in languages like C++, which really just is sugared assembly language, but instead was programmed in a more abstract language like ML.[11] Then the worst that could happen would be an uncaught exception, and nothing really *bad*

[10] I have a nice story about program extraction. The Fundamental Theorem of Algebra formalization that we created was a formalization of an *intuitionistic* proof. Therefore one could extract programs from it (although the proof had not been especially chosen for that.) Now the Fundamental Theorem of Algebra says that every non-constant polynomial has a root. This means that we could extract a *root-finding* algorithm. It took a couple of months to extract the program (there were some bugs in the extraction code), but when we finally got it and ran it, nothing happened. The program just ran and ran without producing any output. Then we tried something simpler. Part of our formalization was the Intermediate Value Theorem, the statement that says that if a function is negative somewhere and positive somewhere else, that it then has a zero in between. (This happens not to be provable intuitionistically: our formalization had an intuitionistic form of it.) By applying this theorem to the polynomial $x^2 - 2$ we extracted a program that calculates $\sqrt{2}$. The output of this program is a stream of fractions that converges to that value. When we ran it, the program *immediately* output the value 0 (every instance of the Intermediate Value Theorem algorithm would do this, regardless of the function that one would put in), and then, after running for hours, it finally output a second 0. However, this second 0 had a smaller limit on how far it was from the desired value of $\sqrt{2}$. There never was a third output. And, although we tried very hard, we never understood what the program was doing all that time.

[11] The ML programming language was *spin off* from the formal methods world. One of the first proof assistants in the world was the LCF system from the seventies. The ML language was at first only developed to program 'tactics' for this system. (In this respect it is similar to the Ltac language for the Coq system that is being developed right now.)

could happen. This is not just something that I would expect. It is my experience that when one programs in ML it might be a bit harder to fit what one wants to do into the framework of the language, but then one needs to spend much less time on debugging. There is an anecdote related to this: when a friend of mine first learned a functional programming language, he decided to try programming in it by 'cloning' his programmable pocket calculator, with all the functions and buttons. It took him quite some time to fit the behavior of this calculator in the type system of the functional language. When he finally succeeded with this, he expected that he would need to debug it for a similar long time. But the program ran, and ran correctly, at the first try.

Another example of what I mean when I say that 'an abstract world view is important' is the example of how one looks at files. In the operating systems of the sixties and seventies files where rather involved objects. One needed to tell the system in some detail where the files would be put on the sectors of the disk, and there would be different *kinds* of files related to how one did this, with names like 'direct' and 'indexed sequential' files. But then with systems like Unix, a file became a much more abstract object. It now just was a sequence of bytes, and how it would be put on the disk was *abstracted away*. This was a huge improvement from the point of view of understanding what was going on. I would like to see more of that kind of abstraction in programming.

For instance, I think it would be a good thing if there would be no integer overflow.[12] I would think that systems would be better behaved if they were programmed in languages like Lisp and Smalltalk where the system automatically switches to 'bignum' behavior when the numbers go outside of the range of machine words. This might sound like a trivial change, but not many programming languages provide this, and wanting this is a good example the 'mind set' that I think will change the way we program.

Let me talk a bit about what research in formal methods currently looks like, and how it currently is applied to get better software.

My impression of the field of formal methods is that most papers are about deductive systems (carrying fancy names like 'ACP_τ', 'the π-calculus', 'STTwU', '$\mathrm{MLW}^{\mathrm{ext}}\mathrm{PU}_{<\omega}$', '$\bar{\lambda}\mu\tilde{\mu}$', etc.) These papers define those systems using pages of deductive rules, and then prove all kinds of things about them and (sometimes) describe experiments with implementations that are based on them. If I am honest, I expect that most of those deductive systems will go nowhere. However, they are part of the *culture* that will produce the few systems that *will* matter, and, like I already said, I expect that this eventually will be very important for software quality.

The technology of formal methods already is being used to prove the correctness of various kinds of software and hardware. Now I think that the word *prove* is a bit misleading here. That suggests *certainty*, and indeed, a proof will give absolute certainty about some property of the program. However *what* should be proved is generally not totally clear. This means that the application of formal

[12] I have been told that some kinds of election fraud make use of the fact that voting machines occasionally fail to notice integer overflow.

methods in computer science, despite its use of 'proofs', does not really give a guarantee that the program will behave as desired, but instead should just be considered to be a rather thorough (and expensive) form of debugging.

To explain why I think that it is not possible to get *total* certainty about the correctness of programs using proofs: consider proving the LaTeX program of Donald Knuth and Leslie Lamport (and after them many others who extended and improved their system) to be correct. What should be the statement that one should prove for this? Suppose that LaTeX puts some text in some document in a wrong font style because of a bug somewhere: how could I have prevented this by proving things about it? The statement that describes how LaTeX should behave (the specification) is of a similar size as the LaTeX source code itself. And it is almost as difficult to get this statement correct as it is to get the program itself correct!

And then, even when the statement that you prove actually is what you want it to be, often assumptions in the specification about the real world will be able to cause trouble. This means that even if you prove that something will not happen, it still might happen. For example, even if you prove that trains will not collide, there is not a *total* guarantee that the trains actually will not do this. I once saw this demonstrated very clearly. There was a demo of an application of formal methods where, as a case study, a little toy train was running on a toy track. The software that drove that toy train had been proved correct. However, the toy train had a toy accident right in front of my eyes, because the sun was shining in its sensors. Apparently the assumptions in the model that had been used in the proof of the safety of this toy train were not satisfied at that time.

I also would like to say something about the relationship between formalization of mathematics and formalization in computer science. There seems to be a trend in formal methods to move to tools that check specific properties of programs (like for instance that variables will not be used when they have not yet been initialized, or that there will be no overflow) without any human intervention, this in contrast with systems like proof assistants that are very general and therefore by necessity interactive. Of course we eventually will need both kinds of system: one should not underestimate the importance of being able to step in and tell the system what to do when the automation does not cut it anymore.

When interactively proving properties of programs, it is important to be able to reason in a 'mathematical' style. Therefore we should have good technology for formalizing mathematical proofs too. I do not see a dichotomy between formalization of mathematics and the applications of formalization in computer science. On the contrary: I think that the thing that is most needed to make formal methods more powerful is the ability to work in a style that is as mathematical as possible.

Luckily, there recently has been much progress in the mathematics that can be formalized. At the start of 2005, formalizations of three famous theorems were finished:

- the Four Color Theorem, using Coq, by Georges Gonthier
- the Prime Number Theorem, using Isabelle/HOL, by Jeremy Avigad

– the Jordan Curve Theorem, using HOL Light, by Tom Hales[13]

(There are two other theorems that have not been formalized yet, but that are always mentioned when people talk about formalization of mathematics. They are:

– the classification of finite simple groups
– Fermat's last theorem

Georges Gonthier told me that he will start working on the first one. And Jan Bergstra put the second one up as a 'grand challenge' for computer science. It will be interesting to see whether anyone will take him up on this challenge.[14])

Nowadays 'big' theorems can be practically formalized, with a lot of work. But on a more mundane scale, the technology also has become good enough to *routinely* formalize ordinary mathematics.

Some time ago I found a 'top hundred' of nice theorems on the web. I decided to investigate how many of them had already been formalized. The result is on:

`http://www.cs.ru.nl/~freek/100/`

Currently three quarters of this list has been formalized, and this fraction is growing fast. The three systems that formalized the most are HOL Light and ProofPower (both are variants of the HOL system) and the Mizar system. Apparently those three systems are currently the best for formalization of mathematics.

In this essay I tried to argue for two things. First, in the field of formalization of mathematics interesting things are happening. And second, developments from the field of formal methods might lead to a culture change in software development that will lead to better software quality everywhere. The first statement I know to be true from my own research experience. The second statement I hope and expect to become true too.

[13] Later in 2005 a formalization of the proof of the Jordan Curve Theorem using the Mizar proof assistant was also finished.

[14] A challenge that already has been taken up is the formalization of the proof by Tom Hales of the 'Kepler conjecture', which states what is the densest way to pack spheres in space. Hales calculates that the formalization of his proof will take twenty years, and calls his project *Flyspeck*.

Towards building better classifiers with ROC analysis

Peter A. Flach, University of Bristol

January 21, 2006

Today, every internet user is familiar with unsolicited bulk email or spam. This is particularly true for computer scientists, who typically have a high web presence and whose email addresses are therefore easily harvested. As an illustration, over the last ten days I received close to 100 spam emails per day. Fortunately, the majority of those spam emails gets filtered out, in my case by a combination of SpamAssassin which runs on the server and adds headers assigning a spam score to each email; and the built-in junk mail filter of Apple Mail. While SpamAssassin uses a hand-crafted rule base to calculate the spam score, Apple Mail uses a Bayesian classifier to recognise spam emails. In a nutshell, Bayesian spam filters work by estimating the probabilities of word w to occur in spam, $P(w|\text{spam})$, and the probabilities of word w to occur in non-spam, $P(w|\text{ham})$, for all words in a dictionary, from a training set. If a new email comes in, the classifier estimates the likelihood of observing this email if it were spam as $\prod_w P(w|\text{spam}) \prod_{w'} (1 - P(w'|\text{spam}))$, where w ranges over all dictionary words occurring in the email and w' over all dictionary words not occurring in the email. Similarly, the likelihood of observing this email if it were non-spam is estimated as and $\prod_w P(w|\text{ham}) \prod_{w'} (1 - P(w'|\text{ham}))$. We then multiply these likelihoods with the appropriate prior probability, $P(\text{spam})$ or $P(\text{ham})$, and predict the class which maximises this product.

It has been shown that this Bayesian decision rule does not necessarily result in the best predictions [6]. The reason is that Bayesian classifiers make unjustifiable independence assumptions (e.g., that occurrence of the words 'Viagra' and 'sex' are independent in spam emails), and therefore the likelihood estimates are not accurate. This doesn't necessarily mean that the likelihood estimates cannot be used for prediction, but it does mean that the decision rule shouldn't only be based on prior probabilities and needs to be learned from the data. This leads us to the problem of how to threshold a continuous value to make binary decisions, which is an old problem originally studied in *signal detection theory*.

In its simplest form, signal detection theory [2, 5] involves the following decision problem. A binary signal with values 0 and d is corrupted by normally distributed noise with zero mean and unit variance. The noisy signal is transmitted to a receiver, who has to reconstruct the original binary signal – that is, the receiver needs to take a decision at each time point whether the signal was absent (0) or present (d), based on a measurement x of the corrupted signal (a number in the interval $(-\infty, \infty)$. This situation can be visualised by means of two Gaussian probability density functions, one centred around 0 (the 'signal absent' Gaussian, denoted $p(x|absent)$) and the other centred around d (the 'signal present' Gaussian, denoted $p(x|present)$). Two sets of data points are sampled from the two Gaussians, and the aim is to reconstruct which point came from which Gaussian (Figure 1). The difficulty of the problem depends on the signal strength d, although it is of course possible that a particular sample is non-overlapping and perfect reconstruction is possible.

In the absence of any other information, most people would probably agree that the most sensible strategy for the receiver is to threshold the received value at $x_0 = d/2$, and to decide that the signal was present if the measurement exceeds this value, and absent otherwise. However, this raises a number of issues, such as: How does the resulting accuracy depend on the signal strength d? What should our strategy be if we don't know d? Should we change the threshold if we know that, e.g., the signal is only present in 10% of the cases? If yes, how? And how do we take into account that a false positive (a decision that the signal was present when in fact it was absent) might be more costly than a false negative (the signal was present but

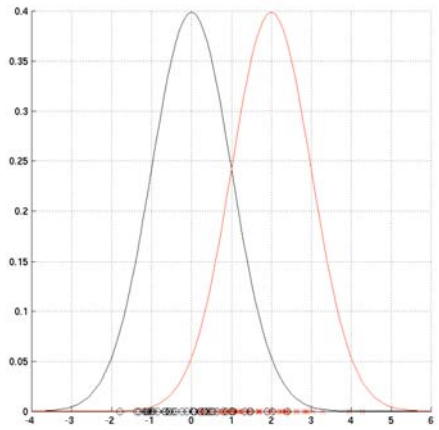

Figure 1: Two unit-variance Gaussians with means at 0 and 2, modelling noisy transmission of a binary signal with values 0 and $d = 2$. Also shown are measurements sampled from each Gaussian ($\times$ indicates 'signal absent' points, and $\circ$ indicates 'signal present' points).

undetected)? How does all this depend on the noise model?

This simple probabilistic model exemplifies a kind of decision problem arising in a range of disciplines from electrical engineering to experimental psychology. The basic idea is that a detection problem can be characterised, for the whole range of possible detection thresholds, by a *Receiver Operating Characteristic* curve or ROC curve. Given a threshold x_0, we can distinguish four quantities:

- the *true positive rate* $tpr = \int_{x_0}^{\infty} p(x|present)dx$;
- the *false positive rate* $fpr = \int_{x_0}^{\infty} p(x|absent)dx$;
- the *false negative rate* $fnr = \int_{-\infty}^{x_0} p(x|present)dx = 1 - tpr$;
- the *true negative rate* $tnr = \int_{-\infty}^{x_0} p(x|absent)dx = 1 - fpr$.

An ROC curve is a plot of true positive rate (percentage of signal present that was detected) against false positive rate (percentage of signal absent that led to incorrect detection), for all possible decision thresholds (see the outermost smooth curve in Figure 2). By construction, an ROC curve is monotonically non-decreasing, starts in $(fpr = 0, tpr = 0)$ corresponding to threshold $+\infty$ (i.e., the signal is always considered absent), and finishes in $(fpr = 1, tpr = 1)$ corresponding to threshold $-\infty$ (i.e., the signal is always considered present). A particular point on the ROC curve is called an *operating point*. The shape of the curve depends on the strength of the signal: the closer the Gaussians are together, the harder it is to detect the signal and the closer the ROC curve will be to the ascending diagonal, which indicates random performance. Conversely, the larger the signal strength d, the easier it is to detect the signal and the closer the ROC curve will be to the step function going through $(fpr = 0, tpr = 1)$. (Equivalently, we could keep the signal strength constant and vary the noise variance.)

The default strategy of setting $x_0 = d/2$ can – in this simple setting – be understood in two ways: (i) as the point where the *likelihood ratio* $\frac{P(x_0|present)}{P(x_0|absent)} = 1$; (ii) as the point where $tpr + fpr = 1$. From the fact that true and false positive rates are obtained by integrating the two Gaussian probability densities, it follows that the likelihood ratio is the slope of the curve in the operating point corresponding to the threshold x_0.

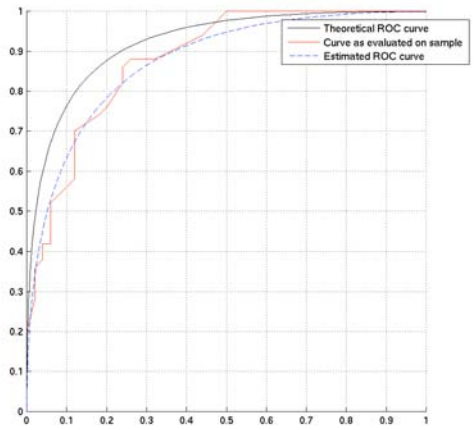

Figure 2: The outermost smooth curve is a theoretical ROC curve, constructed from the underlying model. The other smooth curve is obtained by estimating the means of the two Gaussians from the sample; it is lower than the theoretical ROC curve because the sample means are less far apart than the true means. The jagged curve is obtained by non-parametric estimation, by sliding a decision threshold over the likelihood ratio; it will become smoother (and closer to the theoretical ROC curve) if we increase the size of the sample.

The first strategy thus corresponds to finding the operating point with slope 1, while the second strategy correspond to intersecting the curve with the descending diagonal. In the more general case of non-uniform priors, the first strategy is easily adapted to select the point where the likelihood ratio is equal to $\frac{P(absent)}{P(present)}$. Intuitively, this means that when the signal is more often absent than present we select a point closer to $(fpr = 0, tpr = 0)$, while we get closer to $(fpr = 1, tpr = 1)$ if the signal is more often present than absent. The prior thus tells the decision maker whether it is more important to achieve a low false positive rate, or rather to achieve a high true positive rate. The same mechanism can be exploited to take non-uniform misclassification costs (and possibly non-uniform correct classification profits) into account.

Another way in which the signal detection framework can be generalised is by assuming that the two Gaussians have different variances (for instance, because of additional Gaussian noise in the process generating the signal d). In this case the likelihood ratio will exhibit a local minimum, resulting in a non-convex ROC curve. Figure 3 illustrates this situation. The left figure shows that unequal-variance Gaussians intersect at two points, and hence there are two thresholds where the likelihood ratio is one. The middle figure shows that the ROC curve is non-convex, and that of the two operating points with slope one the left one is better (corresponding to the right-most intersection point on the left figure). The right figure plots the log-likelihood ratio as a function of the threshold, which shows two zero crossings (corresponding to a likelihood ratio of one) with a local minimum in between. Other shapes of ROC curves can be obtained by considering non-Gaussian densities.

The unequal-variance case suggests an alternative decision strategy: to threshold the likelihood ratio rather than the measured signal. In the equal-variance case the likelihood ratio is monotonically increasing with the measurement threshold, which means that the two strategies yield the same result. However, with unequal variances, thresholding the likelihood ratio results in disjoint decision regions. For instance, suppose that we threshold the likelihood ratio at one: Figure 3 shows that the region between the two intersection points is then classified as 'signal absent', while the outer two regions are classified as 'signal present'.

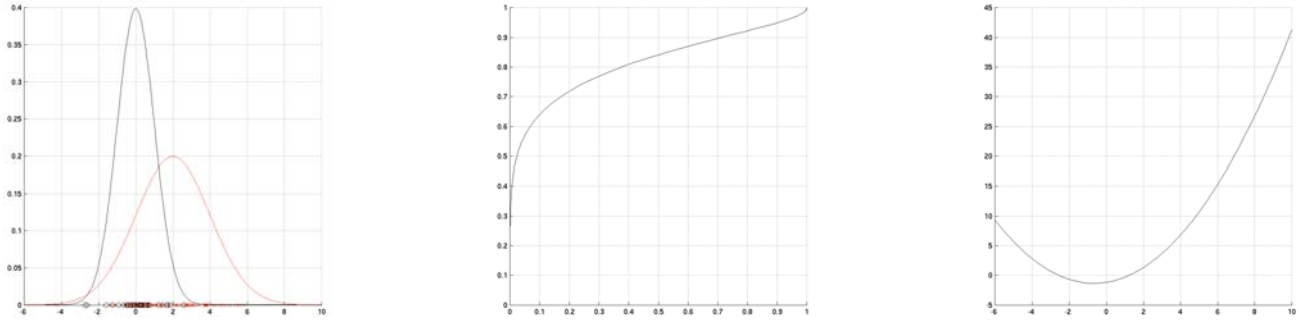

Figure 3: Gaussians with different variances lead to non-convex ROC curves if the decision criterion is a threshold on the measurement value. Left: true distributions and sampled points. Middle: non-convex ROC curve. Right: log-likelihood ratio as a (non-monotonic) function of the threshold.

Consequently, the true and false positive rates are obtained by integrating the Gaussians over these two disjoint regions. It can be shown that, if we threshold the likelihood ratio, the ROC curve is always convex, with the slope in an operating point equal to the likelihood ratio threshold.

This latter strategy of thresholding the likelihood ratio is in fact the dominant one in machine learning and data mining, particularly because of the necessity to deal with multiple independent variables simultaneously. Shifting attention to the type of classification problem prevalent in machine learning and data mining highlights another shortcoming of the basic signal detection framework, which is that the true underlying distributions are needed to construct the ROC curve (for this reason they are sometimes called *theoretical* ROC curves). So let us now assume that all we have are two samples from the two Gaussians modelling the negative and positive class (the $\times$ and $\circ$ points in Figure 1), and we have to estimate the ROC curve. Two strategies immediately suggest themselves:

- parametric estimation: fit two Gaussians to the data by estimating the two class-conditional means and possibly variances, and construct an ROC curve by integrating over the fitted Gaussians;
- non-parametric estimation: slide a decision threshold over the likelihood ratio, and take the relative frequencies of positives and negatives in the positive class region(s) as estimates of the true and false positive rates in the corresponding operating point.

The two methods are illustrated in Figure 2. The parametric method is possible in principle in cases where the learned classifier is parametric (e.g., our Bayesian spam filter), although in the case of multiple independent variables this requires integration over multivariate Gaussians. The non-parametric method is most often used in practice, because it doesn't make any assumptions about the classifier and thus is equally applicable to probabilistic classifiers and to non-parametric models such as decision trees.

For finite samples there are only a finite number of different likelihood ratios that are assigned to test points. So rather than sliding a threshold over the likelihood ratio, it is more efficient to construct the estimated ROC curve from the actual likelihood ratios assigned to test points. The following simple procedure constructs the curve from the test points, ranked on decreasing likelihood ratio: starting in $(fpr = 0, tpr = 0)$, make a step up if the next test point is positive, otherwise make a step to the right for a negative point. In the case of ties, we make the required number of steps up and to the right as one diagonal step. By scaling the vertical step size to one divided by the total number of positives and the horizontal step size to one divided by the total number of negatives, we are guaranteed to end up in $(fpr = 1, tpr = 1)$. Figure 4 shows an ROC

curve thus constructed from a test set with 50 positives and 50 negatives. The number of different likelihood ratios assigned to test points is in fact less than 50, and thus there are a fair amount of ties which manifest themselves as segments longer than the horizontal or vertical step size (this includes all diagonal segments).

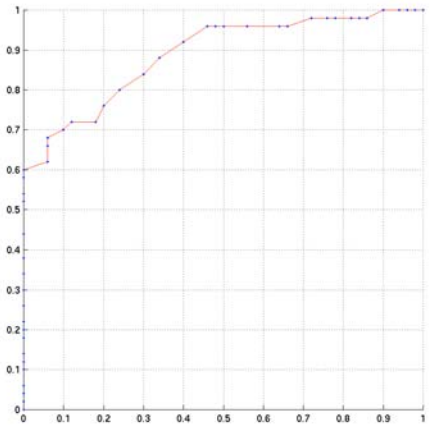

Figure 4: Test set ROC curve: points indicate different likelihood ratios, line segment longer than the step size (including diagonal segments) indicate ties.

It is worth pointing out that for test set ROC curves there is no equivalence between the likelihood ratio as predicted by the classifier and the slope of the curve. This can be clearly seen in Figure 4: whereas the predicted likelihood ratios are – by construction – monotonically decreasing along the curve, the slope is not. Segments of constant slope indicate either ties in the predicted likelihood ratios, or a segment in the ranking where the class distribution is more or less constant, or a combination of both. We can only distinguish between those cases if we don't just plot the ROC curve as a piecewise linear curve, but also plot the points which receive different likelihood ratios. Figure 4 contains several segments, such as the initial vertical segment, the longer horizontal segments to the top right, but also the middle diagonal segment. While the curve contains several distinct points in those segments, it is clear that it would make little or no difference if all test points in the segment would receive the same likelihood ratio. Put differently, while the classifier thinks it makes sense to distinguish points in the segment, the test set demonstrates that this distinction is actually unnecessary and doesn't lead to improved performance. There is reason to believe that a classifier which is actually aware of this, and only assigns different likelihood ratios to points in segments with different slopes, is a better classifier than one which makes unnecessary distinctions. Such a classifier is said to be *well-calibrated* [1]. However, ROC curves provide insufficient support for such an analysis, because the likelihood ratios are only used to construct the ranking and are ignored afterwards. Consequently, two classifiers predicting different likelihood ratios resulting in the same ranking would be characterised by the same test set ROC curve. So an interesting open problem is to extend ROC analysis such that it takes likelihood ratios into account.

To further illustrate the point, consider Figure 5. We have sampled points from two Gaussians with unequal variances, but the classifier assumes unit variance and only estimates the means from the data. Consequently, the test set ROC curve is non-convex; essentially, it can be closely approximated by a 3-segment curve (vertical segment at the start, then a nearly flat segment, followed by another vertical segment). This means that we could discretise the likelihood ratios in three bins without loss of information. The right

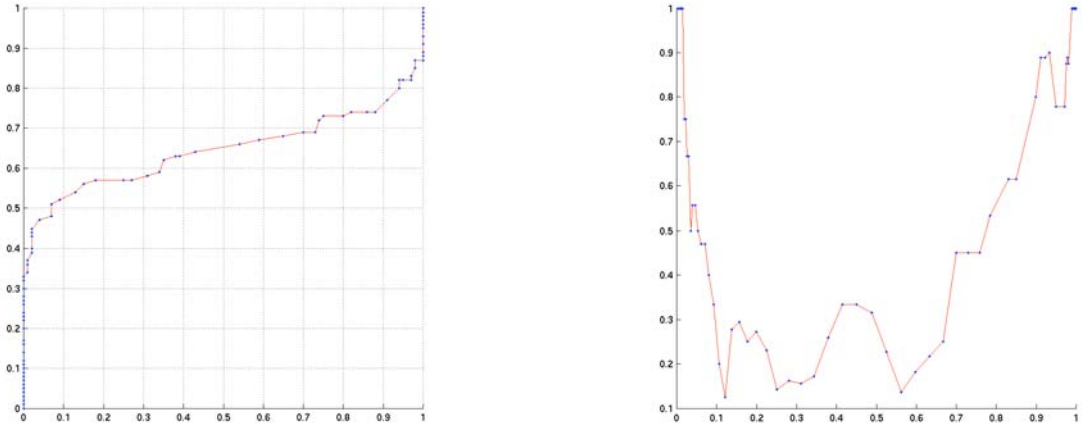

Figure 5: Left: a non-convex test set ROC curve. Right: the slope of the test set ROC curve on the vertical axis against the predicted likelihood ratio on the horizontal axis.

figure plots the slope of the test set ROC curve in an operating point (estimated by sliding a 3-point window over the ranking) against the predicted likelihood ratio in that point. Here, slope and likelihood ratio have been renormalised to the $[0,1]$ interval using the transformation $x \mapsto \frac{x}{x+1}$. Ideally, this *likelihood ratio curve* should stay close to the ascending diagonal – however, the actual curve has two defects: (i) it starts in $(0,1)$ rather than $(0,0)$, corresponding to the vertical segment at the end of the ROC curve; and (ii) it has a plateau below the diagonal in the middle, corresponding to the concavity in the right half of the ROC curve.

The main point of this analysis is to show that ROC curves contain a wealth of information about the behaviour of classifiers. In particular, segments with constant slope indicate locally random behaviour, and two or more segments forming a concavity indicate locally worse than random behaviour. I believe that it is possible to utilise such information to improve models – for instance, we can get rid of the concavity in Figure 5 by discretising the likelihood ratio in **two** bins, one corresponding to the initial vertical segment of the ROC curve, and the other corresponding to the rest. It has recently been shown that we can do better than that, by inverting (rather than collapsing) the ranking in the second segment [4]. Work on such ROC-based model improvement has only recently started, and poses a number of interesting questions.

- Can we incorporate predicted likelihood ratios in the ROC curve, for instance by varying the step size? Or is it better to plot those likelihood ratios explicitly, for instance as done in Figure 5?
- What is a good method for smoothing test set ROC curves in order to estimate slopes? Are there alternatives to the obvious sliding-window approach? How should the window size be chosen? What is the statistical justification for this?
- The shape of the curve is determined by a combination of factors: (i) sampling effects, (ii) estimation effects, (iii) a mismatch between the true model and the fitted model, and (iv) the inherent complexity of the classification problem. For instance, Figure 2 demonstrates both sampling effects and estimation effects, as well as an upper bound on average performance in the form of the theoretial ROC curve. Figure 3 demonstrates a mismatch between fitted and true model, and Figure 5 shows the combined effects of sampling, estimation and model mismatch. Can we come up with a general theory to understand, describe and quantify these effects?

- The number of dimensions in a full ROC analysis is quadratic in the number of classes. How can we generalise the approach to multiple classes while remaining tractable?

ROC analysis already has a considerable impact on current work in machine learning and data mining [3]. Answering these open questions will enable us to exploit its full potential.

References

[1] I. Cohen and M. Goldszmidt. Properties and benefits of calibrated classifiers. In J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi, editors, *PKDD*, volume 3202 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2004.

[2] J.P. Egan. *Signal detection theory and ROC analysis*. Academic Press, New York, 1975.

[3] P.A. Flach. The many faces of ROC analysis in machine learning, 2004. ICML'04 tutorial notes, http://www.cs.bris.ac.uk/ flach/ICML04tutorial/.

[4] P.A. Flach and S. Wu. Repairing concavities in ROC curves. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 702–707. IJCAI, August 2005.

[5] D.M. Green and J.A. Swets. *Signal detection theory and psychophysics*. Wiley, New York, 1966.

[6] N. Lachiche and P.A. Flach. Improving accuracy and cost of two-class and multi-class probabilistic classifiers using ROC curves. In *Proc. 20th International Conference on Machine Learning (ICML'03)*, pages 416–423. AAAI Press, January 2003.

Statuten

Artikel 1.
1. De vereniging draagt de naam: "Nederlandse Vereniging voor Theoretische Informatica".
2. Zij heeft haar zetel te Amsterdam.
3. De vereniging is aangegaan voor onbepaalde tijd.
4. De vereniging stelt zich ten doel de theoretische informatica te bevorderen haar beoefening en haar toepassingen aan te moedigen.

Artikel 2.
De vereniging kent gewone leden en ereleden. Ereleden worden benoemd door het bestuur.

Artikel 3.
De vereniging kan niet worden ontbonden dan met toestemming van tenminste drievierde van het aantal gewone leden.

Artikel 4.
Het verenigingsjaar is het kalenderjaar.

Artikel 5.
De vereniging tracht het doel omschreven in artikel 1 te bereiken door
a. het houden van wetenschappelijke vergaderingen en het organiseren van symposia en congressen;
b. het uitgeven van een of meer tijdschriften, waaronder een nieuwsbrief of vergelijkbaar informatiemedium;
c. en verder door alle zodanige wettige middelen als in enige algemene vergadering goedgevonden zal worden.

Artikel 6.
1. Het bestuur schrijft de in artikel 5.a bedoelde bijeenkomsten uit en stelt het programma van elk van deze bijeenkomsten samen.
2. De redacties der tijdschriften als bedoeld in artikel 5.b worden door het bestuur benoemd.

Artikel 7.
Iedere natuurlijke persoon kan lid van de vereniging worden. Instellingen hebben geen stemrecht.

Artikel 8.
Indien enig lid niet langer als zodanig wenst te worden beschouwd, dient hij de ledenadministratie van de vereniging daarvan kennis te geven.

Artikel 9.
Ieder lid ontvangt een exemplaar der statuten, opgenomen in de nieuwsbrief van de vereniging. Een exemplaar van de statuten kan ook opgevraagd worden bij de secretaris. Ieder lid ontvangt de tijdschriften als bedoeld in artikel 5.b.

Artikel 10.
Het bestuur bestaat uit tenminste zes personen die direct door de jaarvergadering worden gekozen, voor een periode van drie jaar. Het bestuur heeft het recht het precieze aantal bestuursleden te bepalen. Bij de samenstelling van het bestuur dient rekening gehouden te worden met de wenselijkheid dat vertegenwoordigers van de verschillende werkgebieden van de theoretische informatica in Nederland in het bestuur worden opgenomen. Het bestuur kiest uit zijn midden de voorzitter, secretaris en penningmeester.

Artikel 11.
Eens per drie jaar vindt een verkiezing plaats van het bestuur door de jaarvergader-

ing. De door de jaarvergadering gekozen bestuursleden hebben een zittingsduur van maximaal twee maal drie jaar. Na deze periode zijn zij niet terstond herkiesbaar, met uitzondering van secretaris en penningmeester. De voorzitter wordt gekozen voor de tijd van drie jaar en is na afloop van zijn ambtstermijn niet onmiddellijk als zodanig herkiesbaar. In zijn functie als bestuurslid blijft het in de vorige alinea bepaalde van kracht.

Artikel 12.

Het bestuur stelt de kandidaten voor voor eventuele vacatures. Kandidaten kunnen ook voorgesteld worden door gewone leden, minstens een maand voor de jaarvergadering via de secretaris. Dit dient schriftelijk te gebeuren op voordracht van tenminste vijftien leden. In het geval dat het aantal kandidaten gelijk is aan het aantal vacatures worden de gestelde kandidaten door de jaarvergadering in het bestuur gekozen geacht. Indien het aantal kandidaten groter is dan het aantal vacatures wordt op de jaarvergadering door schriftelijke stemming beslist. Ieder aanwezig lid brengt een stem uit op evenveel kandidaten als er vacatures zijn. Van de zo ontstane rangschikking worden de kandidaten met de meeste punten verkozen, tot het aantal vacatures. Hierbij geldt voor de jaarvergadering een quorum van dertig. In het geval dat het aantal aanwezige leden op de jaarvergadering onder het quorum ligt, kiest het zittende bestuur de nieuwe leden. Bij gelijk aantal stemmen geeft de stem van de voorzitter (of indien niet aanwezig, van de secretaris) de doorslag.

Artikel 13.

Het bestuur bepaalt elk jaar het precieze aantal bestuursleden, mits in overeenstemming met artikel 10. In het geval van aftreden of uitbreiding wordt de zo ontstane vacature aangekondigd via mailing of nieuwsbrief, minstens twee maanden voor de eerstvolgende jaarvergadering. Kandidaten voor de ontstane vacatures worden voorgesteld door bestuur en gewone leden zoals bepaald in artikel 12. Bij aftreden van bestuursleden in eerste of tweede jaar van de driejarige cyclus worden de vacatures vervuld op de eerstvolgende jaarvergadering. Bij aftreden in het derde jaar vindt vervulling van de vacatures plaats tegelijk met de algemene driejaarlijkse bestuursverkiezing. Voorts kan het bestuur beslissen om vervanging van een aftredend bestuurslid te laten vervullen tot de eerstvolgende jaarvergadering. Bij uitbreiding van het bestuur in het eerste of tweede jaar van de cyclus worden de vacatures vervuld op de eerstvolgende jaarvergadering. Bij uitbreiding in het derde jaar vindt vervulling van de vacatures plaats tegelijk met de driejaarlijkse bestuursverkiezing. Bij inkrimping stelt het bestuur vast welke leden van het bestuur zullen aftreden.

Artikel 14.

De voorzitter, de secretaris en de penningmeester vormen samen het dagelijks bestuur. De voorzitter leidt alle vergaderingen. Bij afwezigheid wordt hij vervangen door de secretaris en indien ook deze afwezig is door het in jaren oudste aanwezig lid van het bestuur. De secretaris is belast met het houden der notulen van alle huishoudelijke vergaderingen en met het voeren der correspondentie.

Artikel 15.

Het bestuur vergadert zo vaak als de voorzitter dit nodig acht of dit door drie zijner leden wordt gewenst.

Artikel 16.

Minstens eenmaal per jaar wordt door het bestuur een algemene vergadering bijeengeroepen; één van deze vergaderingen wordt expliciet aangeduid met de naam van jaarvergadering; deze vindt plaats op een door het bestuur te bepalen dag en plaats.

Artikel 17.

De jaarvergadering zal steeds gekoppeld zijn aan een wetenschappelijk symposium. De op het algemene gedeelte vaan de jaarvergadering te behandelen onderwerpen zijn

a. Verslag door de secretaris;

b. Rekening en verantwoording van de penningmeester;

c. Verslagen van de redacties der door de vereniging uitgegeven tijdschriften;

d. Eventuele verkiezing van bestuursleden;

e. Wat verder ter tafel komt. Het bestuur is verplicht een bepaald punt op de agenda van een algemene vergadering te plaatsen indien uiterlijk vier weken van te voren tenminste vijftien gewone leden schriftelijk de wens daartoe aan het bestuur te kennen geven.

Artikel 18.

Deze statuten kunnen slechts worden gewijzigd, nadat op een algemene vergadering een commissie voor statutenwijziging is benoemd. Deze commissie doet binnen zes maanden haar voorstellen via het bestuur aan de leden toekomen. Gedurende drie maanden daarna kunnen amendementen schriftelijk worden ingediend bij het bestuur, dat deze ter kennis van de gewone leden brengt, waarna een algemene vergadering de voorstellen en de ingediende amendementen behandelt. Ter vergadering kunnen nieuwe amendementen in behandeling worden genomen, die betrekking hebben op de voorstellen van de commissie of de schriftelijk ingediende amendementen. Eerst wordt over elk der amendementen afzonderlijk gestemd; een amendement kan worden aangenomen met gewone meerderheid van stemmen. Het al dan niet geamendeerde voorstel wordt daarna in zijn geheel in stemming gebracht, tenzij de vergadering met gewone meerderheid van stemmen besluit tot afzonderlijke stemming over bepaalde artikelen, waarna de resterende artikelen in hun geheel in stemming gebracht worden. In beide gevallen kunnen de voorgestelde wijzigingen slechts worden aangenomen met een meerderheid van tweederde van het aantal uitgebrachte stemmen. Aangenomen statutenwijzigingen treden onmiddellijk in werking.

Artikel 19.

Op een vergadering worden besluiten genomen bij gewone meerderheid van stemmen, tenzij deze statuten anders bepalen. Elk aanwezig gewoon lid heeft daarbij het recht een stem uit te brengen. Stemming over zaken geschiedt mondeling of schriftelijk, die over personen met gesloten briefjes. Uitsluitend bij schriftelijke stemmingen worden blanco stemmen gerekend geldig te zijn uitgebracht.

Artikel 20.

a. De jaarvergadering geeft bij huishoudelijk reglement nadere regels omtrent alle onderwerpen, waarvan de regeling door de statuten wordt vereist, of de jaarvergadering gewenst voorkomt.

b. Het huishoudelijk reglement zal geen bepalingen mogen bevatten die afwijken van of die in strijd zijn met de bepalingen van de wet of van de statuten, tenzij de afwijking door de wet of de statuten wordt toegestaan.

Artikel 21.

In gevallen waarin deze statuten niet voorzien, beslist het bestuur.