Nieuwsbrief van de Nederlandse Vereniging voor Theoretische Informatica

Mieke Bruné, Jan Willem Klop, Jan Rutten (redactie)*

Inhoudsopgave

1 Van de Redactie **2**

2 Samenstelling Bestuur **2**

3 Van de voorzitter **2**

4 Theoriedag 2002 **3**

5 Mededelingen van de onderzoekscholen **6**
- 5.1 Institute for Programming research and Algorithmics . . . 6
- 5.2 Dutch Research School in Logic (OzsL), door: Jan van Eijck . . . 8
- 5.3 School for Information and Knowledge systems (SIKS) in 2001 by Richard Starmans 10

6 Wetenschappelijke bijdragen **13**
- 6.1 Towards a type-theoretic interpretation of components: Marcello M. Bonsangue . . 13
- 6.2 Implementations of memory and analogy for language processing: Antal van den Bosch . . . 22
- 6.3 Fingerprints from Quantum Mechanics: Ronald de Wolf . . . 28

7 Ledenlijst **34**

8 Statuten **46**

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

1 Van de Redactie

Beste NVTI-leden,

Graag bieden wij u hierbij het zesde nummer aan van de jaarlijkse NVTI-Nieuwsbrief. Bij het samenstellen hebben we weer de formule van de vorige vijf nummers gevolgd. Zo vindt u naast het programma van de jaarlijkse Theoriedag en de bijgewerkte ledenlijst ook weer enkele bijdragen van collega's met een korte inleiding in hun speciale gebied van expertise. Evenals voorgaande jaren zouden deze Nieuwsbrief en de Theoriedag niet tot stand hebben kunnen komen zonder de financiele steun van onze sponsors: NWO-EW, Elsevier Publishing Company, en de onderzoekscholen IPA, SIKS en OzsL. Namens de NVTI gemeenschap onze hartelijke dank voor deze middelen die ons voortbestaan mogelijk maken! Tenslotte stellen we er prijs op het CWI te bedanken voor de ondersteuning die we genieten, onder andere bij het produceren van de Nieuwsbrief. Ook deze steun is onmisbaar en wordt zeer gewaardeerd!

De redactie,
Mieke Bruné (mieke@cwi.nl)
Jan Willem Klop (jwk@cwi.nl)
Jan Rutten (janr@cwi.nl)

2 Samenstelling Bestuur

Prof.dr. J.C.M. Baeten (TUE)
Dr. H.L. Bodlaender (UU)
Prof.dr. J.W. Klop (VUA/CWI) voorzitter
Prof.dr. J.N. Kok (RUL)
Prof.dr. J.-J.Ch. Meyer (UU)
Prof.dr. G.R. Renardel de Lavalette (RUG)
Prof.dr. G. Rozenberg (RUL)
Prof.dr. J.J.M.M. Rutten (CWI/VUA) secretaris
Dr. L. Torenvliet (UvA)

3 Van de voorzitter

Van de Voorzitter

Ook dit jaar heeft het Bestuur zich beijverd om een interessante Theoriedag te organiseren, met prominente sprekers uit binnen- en buitenland. De Theoriedag zal gehouden worden op vrijdag 1 maart, in het Vergadercentrum La Vie, Utrecht. We hopen en vertrouwen erop dat het programma voor velen van u weer interessant is. Graag tot ziens op 1 maart in Utrecht! Ook zou ik graag van de gelegenheid gebruik willen maken om collega-bestuurslid Grzegorz Rozenberg namens het NVTI bestuur van harte geluk te wensen met het ere-doctoraat van de Technische Universiteit Berlijn dat hij deze maand mocht ontvangen.
Jan Willem Klop, voorzitter NVTI

4 Theoriedag 2002

Vrijdag 1 maart 2002, Vergadercentrum La Vie, Utrecht

Het is ons een genoegen u uit te nodigen tot het bijwonen van de Theoriedag 2002 van de NVTI, de Nederlandse Vereniging voor Theoretische Informatica, die zich ten doel stelt de theoretische informatica te bevorderen en haar beoefening en toepassingen aan te moedigen. De Theoriedag 2002 zal gehouden worden op vrijdag 1 maart 2002, in Vergadercentrum La Vie te Utrecht, op enkele minuten loopafstand van CS Utrecht, en is een voortzetting van de reeks jaarlijkse bijeenkomsten van de NVTI die zeven jaar geleden met de oprichtingsbijeenkomst begon.

Evenals vorige jaren hebben wij een aantal prominente sprekers uit binnen- en buitenland bereid gevonden deze dag gestalte te geven met voordrachten over recente en belangrijke stromingen in de theoretische informatica. Naast een wetenschappelijke inhoud heeft de dag ook een informatief gedeelte, in de vorm van een algemene vergadering waarin de meest relevante informatie over de NVTI gegeven zal worden, alsmede presentaties van de onderzoekscholen.

Programma

09.30-10.00: Ontvangst met koffie
10.00-10.10: Opening
10.10-11.00: Lezing Prof.dr. J. Halpern (University of Cornell)
Titel: Causes and Explanations: A Structural-Model Approach
11.00-11.30: Koffie
11.30-12.20: Lezing Prof.dr. A. Schrijver (CWI, UvA)
Titel: Matching, colouring, scheduling
12.20-12.50: Presentatie Onderzoeksscholen (OzsL, IPA, SIKS)
12.50-14.10: Lunch (Zie beneden voor registratie)
14.10-15.00: Lezing Prof.dr. L.A. Hemaspaandra (University of Rochester)
Titel: Complexity and Lewis Carroll's 1876 Election System
15.00-15.20: Thee
15.20-16.10: Lezing Dr. B. Jacobs (KUN)
Titel: Java Program Verification for Smart Cards
16.10-16.40: Algemene ledenvergadering NVTI

Samenvattingen van de voordrachten

Prof.dr. J. Halpern:
Causes and Explanations: A Structural-Model Approach

What does it mean that an event C "actually caused" event E? The problem of defining actual causation goes beyond mere philosophical speculation. For example, in many legal arguments, it is precisely what needs to be established in order to determine responsibility. (What exactly was the actual cause of the car accident or the medical problem?) Actual causation is also important in artificial intelligence applications. Whenever we undertake to explain a set of events that unfold in a specific scenario, the explanation produced must acknowledge the actual cause of those events.

The philosophy literature has been struggling with the problem of defining causality since the days of Hume, in the 1700s. Many of the definitions have been couched in terms of counterfactuals. (C is a cause of E if, had C not happened, then E would not have happened.) However, all the previous definitions have been shown (typically by example) to be problematic. We propose here new definitions of actual cause and (causal) explanation, using Pearl's notion of structural equations to model counterfactuals. We show that these definitions yield a plausible and elegant account of causation and explanation that handles well examples which have caused problems for other definitions and resolve major difficulties in the traditional account.

This is joint work with Judea Pearl.

Prof.dr. A. Schrijver:
Matching, colouring, scheduling

Finding a maximum matching in a graph belongs to the classical problems in combinatorial optimization, and relates to graph colouring, optimum assignment, and school scheduling. We discuss some new fast methods for these problems, leading to improved complexity bounds.

Prof.dr. L.A. Hemaspaandra:
Complexity and Lewis Carroll's 1876 Election System

It is not enough for an electoral system to have beautiful mathematical properties. In judging systems, one should also be sensitive to the computational complexity issues involved, as one typically wants to be able to, for example, feasibly compute who won an election. In this context, we discuss the complexity of the election system that Lewis Carroll proposed in the year 1876.

This talk is based on joint work with Edith Hemaspaandra and Joerg Rothe.

Dr. B. Jacobs:
Java Program Verification for Smart Cards

The latest generation of smart cards is developed for multiple applications. There is a small operating system on-card, together with a virtual machine, an API and some cryptographic machinery.

These cards are capable of executing small JavaCard programs, called applets, for instance for banking, or communication (GSM). JavaCard is a stripped-down version of Java, designed to run in an environment with limited resources.

Since these cards are especially used for security sensitive applications, correctness is an important issue. In this talk it will be argued that such smart cards form an ideal target (and challenge) for formal methods, because: - the programs that run on these cards are very small, and within the reach of modern verification tools; - these cards are distributed in large numbers, so that flaws can have a huge impact; - the smart card industry is driven by certification demands in which formal methods are important.

This talk will consist of two parts. The first one will describe the work on tool-assisted specification and verification for JavaCard that is being done within a recent European RTD project "VerifiCard" (see www.verificard.org), coordinated by the speaker.

The second part will focus on the contribution of Nijmegen within this project, namely JavaCard API and applet specification, with the specification language JML, for Java Modeling Language (see www.cs.iastate.edu/ leavens/JML.html), and verification with the prooftool PVS (see pvs.csl.sri.com).

Lidmaatschap NVTI

Alle leden van de voormalige WTI (Werkgemeenschap Theoretische Informatica) zijn automatisch lid van de NVTI geworden. Aan het lidmaatschap zijn geen kosten verbonden; u krijgt de aankondigingen van de NVTI per email of anderszins toegestuurd. Was u geen lid van de WTI en wilt u lid van de NVTI worden: u kunt zich aanmelden bij Mieke Bruné (mieke@cwi.nl, CWI), met vermelding van de relevante gegevens (naam, voorletters, affiliatie indien van toepassing, correspondentieadres, email, URL, telefoonnummer).

Lunchdeelname

Het is mogelijk aan een georganiseerde lunch deel te nemen; hiervoor is aanmelding verplicht. Dit kan per email of telefonisch bij Mieke Bruné (mieke@cwi.nl, 020-592 4249), tot een week

tevoren (22 februari)). De kosten kunnen ter plaatse voldaan worden; deze bedragen Euro 10. Wij wijzen erop dat in de onmiddellijke nabijheid van de vergaderzaal ook uitstekende lunchfaciliteiten gevonden kunnen worden, voor wie niet aan de georganiseerde lunch wenst deel te nemen.

5 Mededelingen van de onderzoekscholen

Hieronder volgen korte beschrijvingen van de onderzoekscholen:

- Instituut voor Programmatuurkunde en Algoritmiek
- Landelijke Onderzoekschool Logica
- School for Information and Knowledge systems (SIKS)

5.1 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.

In 2001 the groups Distributed and Embedded Systems (DIES) and Software Engineering (SE) of the University of Twente joined IPA, at all other sites the composition of IPA was unchanged. Researchers from eight universities (University of Nijmegen, Leiden University, Eindhoven University of Technology, University of Twente, Utrecht University, University of Groningen, Vrije Universiteit Amsterdam, and the University of Amsterdam), the CWI and Philips Research (Eindhoven) participated.

In 1997, IPA was formally accredited by the Royal Dutch Academy of Sciences (KNAW) for a period of five years. During 2001, an application for the extension of this accreditation was prepared. As part of this procedure, the spearheads of the IPA research program (Testing, Renovation, Embedded Systems, Software Architecture, Natural Computation and Algorithms for Planning and Design) were discussed and redefined. For the 2002 - 2006 period, IPA will have four so-called "application areas".

Networked Embedded Systems: Embedded systems communicate more and more with each other and with the environment, mostly over the Internet. Hence new issues are becoming important: *Network protocols*, *wireless* technology and *mobility* of devices (as environments become equipped with embedded systems), and *hybrid systems*, the interplay of continuous and discrete systems that occurs when an embedded system controls a continuous environment.

Security: in this very broad area IPA wishes to distinguish itself on at least three technical topics. Correctness of *security protocols* is crucial. *Software security*: the well-established notions and techniques in program correctness need to be extended to include security properties. *Smartcards* form an ideal target for applications of various security techniques because they involve limited resources and are deployed in large number, for often security-critical services.

Intelligent Algorithms: The increasing scope of algorithmic systems in industrial and scientific applications leads to many new requirements. The systems must be always operational, interact with an unpredictable environment, and adapt their behavior over time. The design of intelligent algorithms that deal with these requirements is an important development in current research on discrete algorithms.

Compostional Programming Methods: In software engineering the role of software components is becoming a key issue; unfortunately what a component is or should be is not yet well-understood formally. IPA aims at combining strength in theory and in practice. The leading theme for both sides is compositionality: obtaining larger systems from smaller ones by means of well-understood composition rules. This theme combines design, engineering, validation and verification issues.

Activities in 2001

IPA has two multi-day events per year, which focus on a particular subject. In 2001, the Lentedagen were on Security, the Herfstdagen on Timed Systems.

It has become clear that Security is not an add-on feature, but a fundamental aspect which has to be taken into account all the way through the software development process. As such, it is becoming an extensive research topic, ranging over many traditional areas in computer science. In IPA, security is emerging as a research topic in areas of formal methods, algorithmics, and software technology. It challenges existing techniques and methods, and raises interesting new questions. In industrial practice security is not a recent issue, there a wealth of experience has been acquired in protecting all kinds of applications. The Lentedagen aimed to show both sides of the coin by combining contributions from academia and industry. The program contained sessions on: Cryptography, Verification, Smartcards, Network Control, Content Protection, BAN Logic, Network Security, and Electronic Voting.

Where time has always been a critical factor in the design of computing applications with respect to their efficiency, nowadays it is more often than not also a critical factor in their functionality. In the past decades many untimed formalisms have been adapted to a timed setting and new timed formalisms have been introduced. Examples of such formalisms are temporal logics, timed process algebras, timed automata, and control theory. The complexity of timed systems gives rise to a strong need for the use of advanced, systematic supporting tools and techniques in analysis and verification. The availability of such tools and techniques (e.g. Uppaal, Kronos, RT-Spin, the χ-engine, the PVS toolset) has resulted in many case studies from which experiences can be drawn to further increase our understanding of timed systems. The IPA Herfstdagen presented an overview of the extensive research performed in and around IPA on formalisms, tools, and techniques for the development, analysis, and verification of timed systems. The program was complemented with reports on industrial experiences with timed systems.

In addition to the Lentedagen and Herfstdagen, IPA staged its Basic Course on Software Technology at the University of Utrecht in May and contributed to the 3rd Dutch Model Checking Day and 7th Dutch Testing Day. On the European front, IPA continued its cooperation in the European Educational Forum (EEF) with the research schools BRICS (Denmark), TUCS (Finland), UKII (United Kingdom), IP (Italy), GI (Germany) and FI (France). EEF activities included the Foundations School on Logical Methods in Aarhus (June 25 - July 6, organised by BRICS), the Trends School on Software Architecture in Turku (August 13 -17, organised by TUCS) and the 13th International School on Foundations of Wide Area Network Programming on Lipari Island (July 1-14, organised by IP).

Ph.D. Defenses in 2001

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure.* Faculty of Mathematics and Natural Sciences, UL

R. Ahn. *Agents, Objects and Events, a computational approach to knowledge, observation and communication.* Faculty of Mathematics and Computing Science, TU/e

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle.* Faculty of Science, KUN

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection.* Faculty of Mathematics and Computing Science, TU/e

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA

R. van Liere. *Studies in Interactive Visualization.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA

A.G. Engels. *Languages for Analysis and Testing of Event Sequences.* Faculty of Mathematics and Computing Science, TU/e

J. Hage. *Structural Aspects of Switching Classes.* Faculty of Mathematics and Natural Sciences, UL

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes.* Faculty of Mathematics and Natural Sciences, UL

T.C. Ruys. *Towards Effective Model Checking.* Faculty of Computer Science, UT

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols.* Faculty of Mathematics and Computing Science, TU/e

M.D. Oostdijk. *Generation and presentation of formal mathematical documents.* Faculty of Mathematics and Computing Science, TU/e

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ.* Faculty of Mechanical Engineering, TU/e

D. Bošnački. *Enhancing state space reduction techniques for model checking.* Faculty of Mathematics and Computing Science, TU/e

Activities in 2002

In 2002, the IPA Lentedagen will be held in Heeze from April 3-5. We will have our basic courses on Algorithms and Complexity,(in the first half of the year) and on Formal Methods (in the second half). Further details will become available through our web-site, see:
http://www.win.tue.nl/ipa/activities/.

EEF will organise the Trends School on Massive Data Sets in Denmark in August, and the Foundations School on Specification, Refinement and Verification in Finland in June. In addition, there is the Bertinoro International Summer School for Advanced Studies in Computer Science, 20 - 31 May 2002, in Bertinoro, Italy. For more information on EEF and its activities, see: *http://www.win.tue.nl/EEF/.*

Addresses

Visiting address
Eindhoven University of Technology
Main Building HG 7.22
Den Dolech 2
5612 AZ Eindhoven
The Netherlands

Postal address
IPA, Fac. of Math. and Comp. Sci.
Eindhoven University of Technology
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 http://www.win.tue.nl/ipa/

5.2 Dutch Research School in Logic (OzsL), door: Jan van Eijck

The Dutch Research School in Logic (OzsL) is active in three main areas: mathematical logic, logic in linguistics and philosophy, and logic in computer science. Formal participants in the School are the University of Amsterdam, the Free University in Amsterdam, the University of Utrecht, the University of Groningen, and Tilburg University. In addition, there are numerous associate members, to cater for the need of those who have active scientific links with the OzsL community, while politicial reasons argue against full participation. The general policy of the school is to foster cooperation rather than competition with neighbouring schools, and associate membership is open for all our neighbours.

International cooperation agreements exist with Stanford University, the University of Edinburgh, the University of the Saarland, and the University of Stuttgart. Funding is available for visitor exchanges within this international network, and regular international workshops take place within the network.

The Ph.D. courses offered by the school fall in two categories: courses that are part of the 'school week curriculum', and master classes. School weeks are offered once or twice a year. To give an idea of the contents, here is a recent sample:

Autumn 2001 School Week, Nunspeet The program consisted of the following:

- A one-day opening event where PhD students gave short short accounts of how their research is going: *PHD Accolade* "{".
- Tutorials on Quantified Modal Logic, on Language and Optimality, on Tableau Methods in Theorem Proving, and on Information Retrieval.
- Workshops on Tableau Methods in Theorem Proving, and on Information Retrieval and Language Technology.
- A round table session for students and recent graduates. Topic of discussion: their experiences with the supervision they receive(d).
- A one-day closing event where staff members gave short accounts of their current research interests: *Adult Accolade* "}".

Further details can be found in the web archive of the School, at address
`http://www.ozsl.uva.nl/archive.html`.

The yearly Accolade Event, that always takes place in combination with the Autumn School Week is an occasion where PhD students within the school present their work to the outside world in an informal setting. The purpose of Accolade is to inform the community about how individual Ph.D. projects were progressing, and to stimulate mutual interest. Accolade has as its single aim to inform the Dutch logic community about how the Ph.D. projects within the School are going, irrespective of whether these projects are in an initial, intermediate or final stage of research. Participants and audience are enthousiastic about the formula, where PhD students get useful response in a supportive setting.

Accolade for Adults is a recent addition to the School activities. This complement event to Accolade for PhDs, also known under various silly nicknames, is meant to create an opportunity for staff members within the School to give brief outlines of their research, for an audience consisting of their colleagues and the PhD students of the School.

OzsL issues LIN ('Logic in the Netherlands'), a newsletter for the Dutch logic community (in the broadest possible sense) that appears at rather irregular intervals, both on paper and electronically. Subscription is free of charge; please send an email to the OzsL Office Manager dr Eva Hoogland at `ehooglan@wins.uva.nl` to subsribe. Further information about the school and its activities is available electronically, at http://www.ozsl.uva.nl.

OzsL is deeply involved in European Summer School activities in logic: the well known *ESSLLI* (European Summer School in Logic, Language and Information) meetings, organised under the auspices of FoLLI, the European Association for Logic, Language and Information. The next (14th) ESSLLI School (ESSLLI 2002) takes place in Trento, Italy, 5-16 August, 2002. Further information can be obtained from info@esslli2002 and from the web page http://www.esslli2002.it/. Here is a quote from that web page:

> The main focus of ESSLLI is on the interface between linguistics, logic and computation. The school has developed into an important meeting place and forum for discussion for students, researchers and IT professionals interested in the interdisciplinary study of Logic, Language and Information.

In previous editions of ESSLLI the courses covered a wide variety of topics within six areas of interest: Logic, Computation, Language, Logic and Computation, Computation and Language, Language and Logic. The novelty of the 14th edition is the special emphasis on the interface between the basic areas (Logic, Language, and Computation). So, this edition offers about 50 courses, organised into three interdisciplinary areas (Language & Computation, Language & Logic, and Logic & Computation), at a variety of levels (foundational, introductory, advanced), as well as a number of workshops.

Finally, OzsL collaborates with VvL, the Dutch Association for Logic (Vereniging voor Logica) in organizing activities for a broader community with an interest in logic. See the web site of VvL, at http://www.cwi.nl/vvl.

5.3 School for Information and Knowledge systems (SIKS) in 2001 by Richard Starmans

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. Currently the research program counts 5 research foci: Knowledge Science, Cooperative Systems, Requirements Engineering and Formal specification of IKS, Multimedia, Architectures of IKS. SIKS is an interuniversity research school that comprises 12 research groups in which currently over 225 researchers are active, including 95 Ph.D-students. It's Administrative University is the Free University of Amsterdam. The office of SIKS is located at Utrecht University. SIKS received its accreditation by KNAW in 1998. This year the school will apply for re-accreditation.

Activities in 2001

- Basic course program: 4 basic courses were organized in 2001

  Intelligent Systems (B5) 28 May - 30 May 2001, Conference Center The Queeste, Leusden
  Scientific Director: dr. G.A.W. Vreeswijk (UU)

  Combinatory Methods (B4) 30 May - 1 June 2001 Conference Center The Queeste, Leusden
  Scientific Director: dr. N. Roos (UM)

  System Modelling (B1) 17 December- 19 December 2001 Conference Center Het Bosgoed in Lunteren. Scientific Directors:
  dr. G. Schreiber (UVA)
  prof. dr. R. Wieringa (UT)
  prof. dr. A. de Bruin (EUR)

  Knowledge Modelling (B2) 19 December - 21 December Conference Center Het Bosgoed in Lunteren. Scientific Directors:
  dr. G. Schreiber (UVA)
  prof. dr. R. Wieringa (UT)
  prof. dr. A. de Bruin (EUR)

- Advanced Courses: 3 Advanced courses were organized in 2001

  Full-text Information Retrieval Methods 5 and 6 March 2001, Hotel De Plasmolen, Mook
  Scientific Directors:
  Prof. dr. C. Koster (KUN)

Dr. F. Wiesman (UM)

Computational Intelligence 1 and 2 October 2001, Hotel Mitland, Utrecht
Scientific Directors:
dr. J.C. Bioch (EUR)
dr. I. G. Sprinkhuizen-Kuyper (UM)

Multi Agent systems 12 en 13 November 2001 Hotel Apeldoorn, Apeldoorn Scientific Directors:
dr. C. Witteveen (TUD)
prof. dr. J.-J. Ch. Meyer (UU)
dr. W. van der Hoek (UU)

- Tutorials,workshops, seminars and conferences (co-)organized by SIKS
  - 2 februari 2001: Seminar: the future of Logic, Utrecht (VVL, OzsL, SIKS)
  - 22 februari 2001: Tutorial/Lecture on Machine Learning by Tom Mitchell, Maastricht (UM)
  - 9 april 2001: Tutorial/Lecture on "Web caching and content transformation", Amsterdam (VU)
  - 8 mei 2001: Tutorial/Lecture on "True Interactive Visualization for the Web", Amsterdam (VU)
  - 29 juni 2001: Workshop on eContent-Knowledge Management, Eindhoven (TUE)
  - 12 juli 2001: Design research seminar, Amsterdam (VU) tem 27 september 2001: Masterclass "Modeling and simulating work practice", Amsterdam (UVA)
  - 5 oktober 2001: SIKS-dag 2001 in Het Trippenhuis, Amsterdam
  - 5-6 oktober 2001: Workshop on Reinforcement Learning, Utrecht (UU)
  - 25-26 oktober 2001: Conference: BNAIC 2001, Amsterdam
  - 9 november 2001: Social Event for SIKS-Ph.D.students, Utrecht
  - 23 november 2001: Symposium: Contracts and Coordination, Tilburg (KUB)
  - 30 november 2001: Conference: CLIN2001, Twente (UT)
  - 12 december 2001: Seminar on Groupware/Argumentation, Utrecht (UU)
  - 21 december 2001: Conference: BENELEARN 2001 Antwerpen

Doctoral dissertations

In 2001 eleven researchers obtained their Ph.D.-degree:

- 2001-1 Silja Renooij (UU): Qualitative Approaches to Quantifying Probabilistic Networks.
  Promotores: prof.dr. J.-J.Ch. Meyer (UU), prof.dr.ir. L.C. van der Gaag (UU).
  Co-promotor: dr. C.L.M Witteman (UU).
  Promotie: 12 maart 2001.
- 2001-2 Koen Hindriks (UU): Agent Programming Languages: Programming with Mental Models.
  Promotor: prof. dr. J.-J.Ch. Meyer (UU).
  Co-Promotoren: dr. W. van der Hoek (UU), dr. F.S. de Boer (UU). Promotie: 5 februari 2001.

- 2001-3 Maarten van Someren (UvA): Learning as problem solving.
  Promotor: prof. dr. B.J. Wielinga (UvA). Promotie: 1 maart 2001.
- 2001-4 Evgueni Smirnov (UM): Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets.
  Promotor: Prof. dr. H.J. van den Herik (UM/RUL).
  Promotie: 22 februari 2001.
- 2001-5 Jacco van Ossenbruggen (VU): Processing Structured Hypermedia: A Matter of Style.
  Promotor: prof.dr. J.C. van Vliet (VU).
  Promotie: 10 april 2001.
- 2001-6 Martijn van Welie (VU): Task-based User Interface Design.
  Promotor: prof.dr. J.C. van Vliet (VU).
  Co-promotores: dr. G.C. van der Veer (VU), dr. A. Eliëns (VU).
  Promotie: 17 april 2001.
- 2001-7 Bastiaan Schonhage (VU): Diva: Architectural Perspectives on Information Visualization. Promotor: prof. dr. J.C. van Vliet (VU).
  Copromotor: dr. A. Eliëns (VU).
  Promotie: 8 mei 2001.
- 2001-8 Pascal van Eck (VU): A Compositional Semantic Structure for Multi-Agent Systems Dynamics.
  Promotores: prof.dr. F.M.T. Brazier (VU), prof.dr. J. Treur (VU).
  Promotie: 12 juni 2001.
- 2001-9 Pieter Jan 't Hoen (RUL): Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes.
  Promoter: prof. dr. G. G. Engels.
  Copromotoren: dr. L.P.J. Groenewegen, dr. P.W.M. Koopman.
  Promotie: 25 oktober 2001.
- 2001-10 Maarten Sierhuis (UvA): Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design.
  Promotores: Prof. Dr. R. de Hoog (UvA), Prof. Dr. B. Wielinga (UvA).
  Co-promotor: Dr. W.J. Clancey (IHMC/NASA Ames Research Center).
  Promotie: 28 september 2001.
- 2001-11 Tom M. van Engers (VUA): Knowledge Management: The Role of Mental Models in Business Systems Design.
  Promotor: Prof.dr. J.M. Akkermans (VUA).
  Co-promotor: Dr. G.C. van der Veer (VUA).
  Promotie: 11 december 2001.

6 Wetenschappelijke bijdragen

6.1 Towards a type-theoretic interpretation of components: Marcello M. Bonsangue

Leiden Institute of Advanced Computer Science, email: marcello@liacs.nl

Abstract

Many techniques, programming methodologies and programming languages have been developed in the last years to ease a structured and reusable design of software systems. While many convincing programming methodologies, like object-orientation, support well modern programming tasks, like distribution and mobility, they do not seems to have an equal support of modern programming requirements, like distributed and independent extensibility of reusable piece of software. In this paper we briefly introduce component-oriented programming as a natural evolution of object-oriented concepts and sketch an interpretation of a component model in terms of the polymorhic λ-calculus with recursive types and subtyping.

The research of Dr. Bonsangue has been made possible by a fellowship of the Royal Netherlands Academy of Arts and Sciences

Introduction

Software systems have become during the years more and more complex. Building software systems by integrating pre-existing software components has been the dream of software engineers since a long time [McI68], but only in the last decade technologies have appeared which make possible the construction of software systems by assembly reusable components. This design methodology is often called component-based development or component-based programming.

Component-oriented programming incorporates successful concepts from established paradigms like object-orientation while trying to overcome some of their deficiencies. The emphasis in component-based systems lies on *encapsulation* of software structures as components, *interfaces* as mechanism for abstracting from the component implementation, and *polymorphims* as mechanism for allowing different implementations to be bound at run-time to the same component type (dictated by its interfaces). Inheritance is not as fundamental as in object-oriented programming since it has been recognized as one of the sources of weakness of the object-oriented paradigm [Sny86]. Thus components offer a natural unit for distribution and abstraction, hiding and structuring the complexity of large distributed programs. They can be reused across many applications and may support variability and adaptability through parametric interfaces.

Despite the worldwide recognition for the need of a more systematic reuse and structuring of software, as indicated by the breadth and depth of the interest generated by existing component-based technologies like Microsoft's Component Object Model [COM], Sun's Enterprise JavaBeans [EJB] and OMG's CORBA [COR], components lack a clear mathematically well-understood formal foundation. In this paper we present an interpretation of component-oriented programming in term of a typed λ-calculus with subtyping. This interpretation is based on previous encoding of objects as typed λ-terms, confirming the vague intuition that components are proper generalization objects, and hopefully clarifying their differences.

We will proceed as follows. In the next section we informally introduce object- and component-oriented programming. In Section 6.1 we briefly present a component model inspired to Microsoft's Component Object Model [COM], and in Section 6.1 we propose its type-theoretical interpretation as an encoding in a typed polymorphic λ-calculus with subtyping. It is thus not a direct presentation of typing rules for a specific component language, but rather a component-oriented programming style in typed λ-calculus, in the spirit of several object encodings from the literature [Car84, Coo89, PT94, ACV96].

From structured to component-oriented programming

Software systems provide infrastructure in virtually all industries today. Over the past decade, these systems have become very large and often distributed among several computing platforms, and at the same time they have gained an increasingly important role within our society. For managing and reducing the complexity of those systems, various problem solving techniques and design methodologies have been proposed over the years. The traditional *top-down programming* approach advocated in 1970s [Dij72] was based on the principle of divide and conquer: break a complex problem into smaller pieces, and solve those pieces to obtain a solution for the original problem. This approach has focus mainly on functional decomposition, typically by considering simple data structures invariant under refinements of programs.

As the complexity of systems increased, the production of high-quality programs has become more difficult and expensive, so that the reuse of the existing pieces of software has grown in importance. The *object-oriented programming* approach promoted in the 1990's [Boo91] improved the top-down programming approach by allowing both functionality and data representation to be refined in the design process. The fundamental principles underpinning object-oriented technology are:

- objects are bundling of data and functions manipulating those data;
- objects hide detailed information that is irrelevant at higher levels of abstraction;
- each object has a system-wide unique identity.

Information hiding is obtained through well-defined interfaces describing the assumptions that objects can make about each others [Par71]. Typically object interfaces describe the *provided* assumption, that is, the services offered by an object to its environment. Encapsulation of data and functions together in objects with well-defined interfaces increase modularity, and encourages design of systems with low coupling (the number of times a change in one module necessitates change in another module) and high-cohesion (the level of abstraction described by the interfaces).

Software modules with high cohesion and low coupling may be *reused* in another system and can be easily replaced in the existing system [SP00]. As such, object-oriented programming supports two types of reusability: reuse between systems and reuse within a system. Further reuse within a system is realized with inheritance: new objects are derived from old by reusing old functionalities and adding new ones. A consequence of (method) reuse is reflected by the notion of *self*, a special name used by a method to refer to its host object. Without it, reuse would be more limited. Besides reuse, object-oriented programming support easing maintenance by dynamic binding: the object executing a function is determined at run-time.

Today software systems are distributed and often without any centralized control (as with electronic commerce systems). Software components must be easily replaceable either by a completely different implementation of the same functions, or by adding new functionality or by an upgraded version of the current implementation. Recently, trends have emerged for software development through the planned integration of pre-existing software components. This is often called *component-oriented programming.* The primary objectives of component-oriented programming are reuse and *independent extensibility.* A software system is consider to be *extensible* only if it has mechanism supporting the addition of new functionality, and is said to be *independently extensible* if it is extensible and if independently developed extensions can be combined [Szy98].

Component-oriented programming incorporates successful concepts from established paradigms, like object-orientation, while trying to overcome some of their deficiencies. In particular, components adhere to the above fundamental principles of object-orientation and extend them by guaranteeing a stronger notion of encapsulation: components are black-boxes whose boundary cannot be crossed. This stronger notion of encapsulation has two significant implications. First, it supports component implementation by other paradigms than object-orientation. Second, implementation inheritance has to be avoided, at least in an undisciplined way. Indeed, while inheritance in object-orientation is the main mechanism to support extensibility, it breaks encapsulation [Sny86].

Consider the following example, known as the fragile class problem (in the present simple form taken from [FF01]). An object O has three operations *add*, *rem* and *mrem* for adding, removing and removing several elements at once from some data structure. We want to define an object O' that inherits all operation from O and also support an operation *size* for querying the number of elements in the data structure. However we cannot implement O' without knowing the implementation details hidden in O: if O implements *mrem* by calling *rem* repeatedly, we have to override *rem* in O' to correctly implement *size*, whereas if *mrem* does not call *rem*, we have to override *mrem* instead. In object-orientation this problem is typically solved by using design conventions [MS98].

Another concept component-oriented programming shares with object-orientation is that of *polymorphism*. Polymorphims in the form of structural subtyping (i.e., subtyping determined by the type structure) supports extensibility because if A is a subtype of B, then any component of type A may be used without type error in any context that requires a component of type B. The type of a component is thus seen as a contract describing the services offered by a system to its environment [Mey88]. Of course, this contract focus only on the syntactic aspects of the services provided, and not on their behavior (an aspect captured by behavioral subtyping [Ame91, LW94]), and other quality of services like time and space guarantees (as specified, for example, by QDL, a Quality Description Language [LS+98]).

A component model

In this section we informally present our component model. In the previous section we have discussed *why* components are needed for today software, next we proceed by describing *what* a component is, and *how* components interact.

A component consists of a state and an identity. All interactions with a component is through its labeled interfaces. As with object, an interface consists of a collection of differently named abstract operations (messages), each with a defined type. To avoid corruption of encapsulation, implementation inheritance across components is not supported.

A component can support any number of interfaces, so to allow for independent extensibility. For example, a system maintainer, who need not to be the designer of the initial component, can add new interfaces in order to include new functionalities. When an interface is added to a component a new fresh name is assigned to it, local to the component. In this way name conflicts are avoided, and a two equally named messages with the same type can be bound to the same component [FF01].

In traditional object-oriented models, name conflicts do not arise because objects provide a state and an implementation of all the messages of one single interface (thus forbidding extensibility). Modern object-oriented languages, like Java, allow for some extensibility but not for a distributed one. Indeed one single object may implement many different interfaces, but the messages in all the interfaces must have either a different name or a different type.

Summarizing, the major difference between the above component model and a standard object-oriented model are: any number of interface per component, labeled interfaces, and no implementation inheritance).

Components can be created dynamically, and interact by message invocation. Messages are bound to a component, but belong to a specific interface. Informally, we mean by $c \Leftarrow i.m$ the invocation by a client of a message m that belong to the interface named i of a component c. This all look a simple extension of the object-oriented dynamic lookup mechanism. However interface names are assigned to the component interfaces dynamically, when the component is extended. As a consequence, a client may know the (sub)type of a component with a specific interface, but not the interface name needed to invoke a specific message from that interface. To learn interface names clients have the possibility to enquire a components via a known standardly named interface providing operations that return a name for a given signature.

The encoding

In this section we formalize the above component model by encoding it into a second-order polymorphic λ-calculus: $\mathbf{F}_{<:\mu}$ calculus. We first introduce the syntax of the calculus with some informal explanation, and then define our encodings using a simple example.

The $\mathbf{F}_{<:\mu}$ calculus

In the last decade, a number of typed λ-calculi have been proposed as foundation for typed object oriented programming. An overview can be found in [BCP99]. The encoding in this paper is expressed in the $\mathbf{F}_{<:\mu}$ calculus [CW85]. Basically, it is a second-order polymorphic λ-calculus with subtyping, existential types, recursively defined types, recursive functions, and records. Types and terms have the following syntax, respectively:

T	$::=$	X	type variable
		Bool	Boolean type
		Int	Integer type
		Top	maximal type
		$\{l_1{:}T, \ldots, l_n{:}T\}$	record type
		$T \rightarrow T$	function type
		$\forall(X <:T)T$	universal type
		$\exists(X <:T)T$	existential type
		$\mu(X)T$	recursive type

t	$::=$	x	variable
		b	boolean expression
		e	integer expression
		$\lambda(x{:}T)t$	abstraction
		$t(t)$	application
		$\{l_1 = t, \ldots, l_n = t\}$	record construction
		$t.l$	record field application
		$\lambda(X <:T)t$	type abstraction
		$t(X)$	type application
		$\mathsf{pack}\ [X <:T, t]\ \mathsf{as}\ T$	existential packaging
		$\mathsf{open}\ t\ \mathsf{as}\ [X, x]\ \mathsf{in}\ t$	existential unpackaging
		$\mathsf{let}\ x = t\ \mathsf{in}\ t$	local definition
		$\mathsf{letrec}\ x(x{:}T){:}T = t\ \mathsf{in}\ t$	recursive local definition

As usual, $A <:B$ means that A is a subtype of B, and $a{:}A$ means that a is a well-formed term of type A. For simplicity we will omit all typing and subtyping rules, that can be found, for example, in [ACV96]. For instance, it holds that if $a{:}A$ and $A <:B$ then $a{:}B$. An important rule states that a record type R is a subtype of R' if R has at least all the field names of R' and moreover the types associated with those names in R are a subtype of the types associated with the respective name in R'.

The basic types are Boolean, integer, function types and record types. The latter are a list of differently named field and of the types of the associated values. Given a term a of record type, the value of the field labeled l in a can be obtained by $a.l$.

Existential and universal types are standard from abstract data types [MP88]. Informally, the term pack $[X <:A, u]$ as U creates a term of type $\exists(X <: A)B$, where $U <:A$ and u is of type $B[U/X]$, thus hiding the real type of u and revealing only partial information about its type through A. Conversely, given a term u of existential type, the term open u as $[X, x]$ in t yields binding in t for the type variable X and the variable x, denoting the 'representation' type of u and its 'inside'.

The recursive type $\mu(X)T$ denotes a solution of the type equation $X = T$, where X can occur free in T. For simplicity, we omit the terms denoting the isomorphism between $\mu(X)T$ and its unfolding $T[\mu(X)T/X]$.

The term let $x = a$ in b denotes the use of a term a in the term b through the local variable x. Similarly, the term letrec $x(y{:}A){:}B = b$ in c denotes the use in the term c of a a recursive definition of a function x of type $A \to B$.

We assume standard definition of reduction and conversion and write $a =_\beta b$ to denote that a and b converge to the same set of terms up-to α conversion.

A functional component model

We represent an interface as a type operator [BCP99]. For example,

$$\begin{aligned} I1(X) &= \{get{:}\mathsf{Int}, set{:}\mathsf{Int} \to X\} \\ I2(X) &= \{set{:}\mathsf{Int} \to X, double{:}X\} \end{aligned}$$

are two interfaces of a cell component. The first with two messages: *get* which intended meaning is to return the cell current content, and *set* which sets the cell to the integer provided as parameter. The second interface has also two messages: *set* which sets the cell to the result from the addition of the of the current content and the integer provided as parameter, and *double* which sets the cell to the double of its current value. Note the name conflict between the two *set* messages: the same signature but different intended meaning (it would have been better to name *add* the *set* message of the second interface, but in a independent extension this could not be guaranteed). Of course, an interface does not guarantee a specific meaning to its messages, as they are only abstract operations.

Intuitively, the variable X plays the role of the "type of self", that is, the type of the special name *self* used by an implementation of the interface to refer to itself, for example for invoking other operations of the same implementation.

Subtyping is playing a prominent role in our encoding. It is related to the notion of extensibility and refinement: a term a of type A can be replaced by any other term of type $B <: A$, at least from a type theoretical point of view. For a component A subtype of a component B this implies support for the same and more interfaces of B.

Next we come to the encoding of a component C with interfaces $I_1, \ldots, I_n$. Its type, denoted by $C(I_1, \ldots, I_n)$, is defined as follows:

$$C(I_1, \ldots, I_n) \quad = \quad \mu(X)\exists(Y <:X)\{i_1 = I_1(Y), \ldots, i_n = I_2(Y)\}$$

where $i_1, \ldots, i_n$ are distinct integers, standing for the local names of the component interfaces. In the implementation the type of Y will be the actual type of the component. However this is hidden to the environment so to allow for future extensions without the environment having knowledge of them. Indeed, by unfolding the above type we have that

$$C(I_1, \ldots, I_n) = \exists(Y <:C(I_1, \ldots, I_n))\{i_1 = I_1(Y), \ldots, i_n = I_2(Y)\}$$

In other words, the actual hidden type of a component can be any subtype of the type it reveals.

For example, a component implementing the two cell interfaces $I1$ and $I2$ above can be defined

as the following term:

$$
\begin{array}{rcl}
mycell & = & \textsf{letrec}\ new(s{:}\{x{:}\textsf{Int}\}){:}C(I1, I2) = \\
& & \quad \textsf{pack}\ [X <: C(I1, I2), \\
& & \quad\ \{i_1 = \{get = s.x, set = \lambda(n{:}\textsf{Int})new(\{x = n\})\}, \\
& & \quad\ \ i_2 = \{set = \lambda(n{:}\textsf{Int})new(\{x = s.x + n\}), double = new(\{x = s.x * 2\})\} \\
& & \quad\ \ \}]\ \textsf{as}\ C(I_1, I_2) \\
& & \quad \textsf{in}\ new(\{x = 1\}) \\
& & {:}C(I_1, I_2)
\end{array}
$$

Let us denote by $c \Leftarrow i.m$ the sending of a message m of type A belonging to an interface named i of a component c. Message sending is encoded as follows

$$c \Leftarrow i.m \quad = \quad \textsf{open}\ c\ \textsf{as}\ (X, r)\ \textsf{in}\ r.i.m{:}A$$

The term executing the message sending $c \Leftarrow i.m$ needs not to be aware of the type of the component c, say $C(I_1, \ldots, I_n)$. Assuming I_k is the type operator corresponding to the interface named i in c, we have $C(I_1, \ldots, I_n)\ <:\ C(I_k)$ and hence $c{:}C(I_k)$.

In our cell component, we have for example that $(mycell \Leftarrow i_2.double) \Leftarrow i_1.get$ reduces to 2:

$$
\begin{array}{ll}
\multicolumn{2}{l}{(mycell \Leftarrow i_2.double) \Leftarrow i_1.get \ =_\beta\ (new(\{x = 1\}) \Leftarrow i_2.double) \Leftarrow i_1.get} \\
=_\beta & (\textsf{pack}\ [X <: C(I1, I2), \\
& \quad \{i_1 = \{get = \{x = 1\}.x, set = \lambda(n{:}\textsf{Int})new(\{x = n\})\}, \\
& \quad\ \ i_2 = \{set = \lambda(n{:}\textsf{Int})new(\{x = \{x = 1\}.x + n\}), double = new(\{x = \{x = 1\}.x * 2\})\} \\
& \quad \}]\ \textsf{as}\ C(I1, I2) \Leftarrow i_2.double) \Leftarrow i_1.get \\
=_\beta & (\{i_1 = \{get = \{x = 1\}.x, set = \lambda(n{:}\textsf{Int})new(\{x = n\})\}, \\
& \quad i_2 = \{set = \lambda(n{:}\textsf{Int})new(\{x = \{x = 1\}.x + n\}), double = new(\{x = \{x = 1\}.x * 2\})\} \\
& \ \} \Leftarrow i_2.double) \Leftarrow i_1.get \\
=_\beta & (\{set = \lambda(n{:}\textsf{Int})new(\{x = \{x = 1\}.x + n\}), \\
& \quad double = new(\{x = \{x = 1\}.x * 2\})\}.double) \Leftarrow i_1.get \\
=_\beta & new(\{x = 2\}) \Leftarrow i_1.get \\
=_\beta & \textsf{pack}\ [X <: C(I1, I2), \\
& \quad \{i_1 = \{get = \{x = 2\}.x, set = \lambda(n{:}\textsf{Int})new(\{x = n\})\}, \\
& \quad\ \ i_2 = \{set = \lambda(n{:}\textsf{Int})new(\{x = \{x = 2\}.x + n\}), double = new(\{x = \{x = 2\}.x * 2\})\} \\
& \quad \}]\ \textsf{as}\ C(I1, I2) \Leftarrow i_1.get \\
=_\beta & \{i_1 = \{get = \{x = 1\}.x, set = \lambda(n{:}\textsf{Int})new(\{x = n\})\}, \\
& \ i_2 = \{set = \lambda(n{:}\textsf{Int})new(\{x = \{x = 1\}.x + n\}), double = new(\{x = \{x = 1\}.x * 2\})\} \\
& \ \} \Leftarrow i_1.get \\
=_\beta & \{get = \{x = 2\}.x, set = \lambda(n{:}\textsf{Int})new(\{x = n\})\}.get \\
=_\beta & \{x = 2\}.x \\
=_\beta & 2
\end{array}
$$

In our encoding, a component sending a message is required to know the label associated with the interface the message belongs to. Recall that interface names are local to a component and assigned to interfaces dynamically, when added to the component. Next we explain how a component can learn those names. It is sufficient to assign to every component a common interface, say

$$I0(X) \quad = \quad \{iget{:}\textsf{Top} \to \textsf{Int}\}$$

including a message *iget* which intended meaning is to return the integer used as name for the interface generating the type passed as parameter. For example, in our cell component we could implement *iget* by the following second-order function

$$
\begin{aligned}
iget \;=\; & \lambda(X <: Top)\ \textsf{if}\ X <: C(I0)\ \textsf{then}\ 1\ \textsf{else} \\
& \textsf{if}\ X <: C(I1)\ \textsf{then}\ i_1\ \textsf{else} \\
& \textsf{if}\ X <: C(I2)\ \textsf{then}\ i_2\ \textsf{else}\ 0
\end{aligned}
$$

where $\textsf{if} - \textsf{then} - \textsf{else}$ is the obvious conditional function, 1 is the integer assigned as standard name for $I0$ to all components, and 0 is the integer returned in case the interface passed as input is not supported by a component. Using this implementation it is easy to see that

$$mycell \Leftarrow (mycell \Leftarrow 1.iget(C(I1))).get$$

reduces to 1 (basically because $(mycell \Leftarrow 1.iget(C(I1))$ reduces to i_1).

Conclusion

We have introduced the general ideas behind component-oriented programming, and presented a simple component models based on Microsoft's COM [COM]. We have also presented the sketch of an interpretation of this model into a typed λ-calculus. Our interpretation uses techniques originally developed for object-calculi by Abadi, Cardelli and Viswanathan [ACV96, AC96], like the combination of recursive and existential type to encode 'extensible' objects.

Related work in formalizing type systems for components include the work of Ibrahm and Szypersky on the language COMEL [IS97], based on a syntactic formalization of COM. Our approach is more general, as it formalize a general model for independently extensible systems, rather rather than a specific syntax-directed component model.

For simplicity we restricted ourself only to a type theoretical treatment of components, and have not treated other important structural and behavioral issues in component-oriented programming, like require interfaces (i.e. the services required by a component form the environment) and component behavioral specification. Those aspects can be, for example, modeled by using a formal-logic-based component interface description language that conveys the observable semantics of components [ABB00].

Future work include a domain theoretical interpretation for components, as well as the integration of specification and verification techniques with this type framework.

Acknowledgements We like to thank Farhad Arbab, Frank de Boer, Juan Scholten, Joost Kok and Willem-Paul de Roever for stimulating discussions about component (formal) models. Thanks also to Martin Steffen for discussions about system $\mathbf{F}_{<:}$.

References

[ACV96] M. Abadi, L. Cardelli, and R. Viswanathan. An interpretation of objects and object types. In *Principle of Programming Languages*, pages 296-409, 1996.

[AC96] M. Abadi and L. Cardelli. *A Theory of Objects.* Springer-Verlag, 1996.

[Ame91] P. America. Designing an object-oriented programming language with behavioural subtyping. In *Proceeding of the REX workshop "Foundations of Object-Oriented Languages"*, volume 489 of LNCS, pages 60–91, Springer-Verlag, 1991.

[ABB00] F. Arbab, M.M. Bonsangue, and F.S. de Boer. A logical interface description laguage for components. In *Proc. of the 4th Int. Conference on Coordination Models and Languages* volume 1906 of LNCS, pages 249-266. Springer-Verlag, 2000.

[BCP99] K.B. Bruce, L. Cardelli, and B.C. Pierce. Comparing Object Encodings. In *Information and Computation*, 155:108–133, 1999.

[Boo91] G. Booch. *Object-Oriented Design with Applications*, Benjamin Cummings Pub. Co., 1991.

[Car84] L. Cardell. A semantics of multiple inheritance. In *Information and Computation*, 76(2/3):138–164, 1988.

[COM] Microsoft Corporation. Available at `http://www.microsoft.com/COM`.

[Coo89] W. Cook. *A Denotational Semantics of Inheritance*. PhD thesis, Brown University, 1989.

[COR] Object Management Group. CORBA. Available at `http://www.omg.org`.

[CW85] L. Cardelli and P. Wegner. On understanding types, data abstraction and polymorphism. In *Computing Surveys*, 17(4), 1985.¡

[Dij72] E.W. Dijkstra. Notes on structured programming. In O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, editors, *Structured Programming*. Academic Press, 1972.

[EJB] SUN Microsystem. Enterprise JavaBeans. Available at `http://java.sun.com/products/ejb`.

[FF01] P.H. Fröhlich and M. Franz. On Certain Basic Properties of Component-Oriented Programming Languages (Position Paper). In D.H. Lorenz and V.C. Sreedhar, editors, *Proceedings 1st OOPSLA Workshop on Language Mechanisms for Programming Software Components*, Technical report NU-CCS-01-06, pages 15–18, College of Computer Science, Northeastern University, Boston 2001.

[IS97] R. Ibrahim and C. Szyperski. The COMEL Language. Technical Report FIT-TR-97-06, Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia.

[LS+98] J.P. Loyall, R.E. Schantz, J.A. Zinky, and D.E. Bakken. Specifying and measuring quality of service in distributed object systems. In *Proceedings of the 1st International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '98)*, IEEE Computer Society, 1998.

[LW94] Liskov, B. H. and J. M. Wing. A behavioral notion of subtyping. *ACM Transaction on Programming Languages* 16:6, ACM Press, 1994

[McI68] M.D. McIlroy. Mass-produced software components. In *Proceedings of the NATO Software Engineering Conference*, pages 138–155, 1968.

[MP88] J.C. Mitchell and G.D. Plotkin. Abstract types have existential type. In *ACM Trans. on Programming Languages and Systems*, 10(3):470–502, 1988.

[MS98] L. Mikhajlov and E. Sekerinski. A study of the fragile base class problem. In *Proceedings ECOOP'98*, LNCS, Springer-Verlag, 1998.

[Sny86] A. Snyder. Encapsulation and inheritance in object-oriented programming. In M. Meyrowitz, editor, *Object-Oriented Programming, Systems, Languages and Applications*, pages 38–45, SIGPLAN Notices 21:11, 1986.

[Mey88] B. Meyer. *Object-Oriented Software Construction*, Prentice-Hall, 1988.

[SP00] P. Stevens and R. Pooley. *Using UML: Software Engineering with Objects and Components*, updated edition, Addison-Wesley, 2000.

[Szy98] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley, 1998.

[Par71] D. Parans. Information distribution aspects of design methodology. In *Proceedings of the 1971 IFIP Congress*. North Holland Publishing, 1971.

[PT94] B. Pierce and D. Turner. Simple type-theoretic foundation for object-oriented programming. In *Journal of Functional Programming*, 4:207-247, 1994.

6.2 Implementations of memory and analogy for language processing: Antal van den Bosch

Tilburg University

One of the sections within the Faculty of Arts at Tilburg University focuses its research programme on the computational aspects of natural language processing. This contribution focuses on one strand of research of the section, bundled within the "Induction of Linguistic Knowledge" (ILK) research group. The work of this group is built on the hypothesis that machine learning methods from AI, which offer tools for implementing memory and analogy, can be used successfully to learn and model complex natural language processing tasks.

The ILK Research Group was founded by prof. Walter Daelemans during the early '90s. It has been funded through a number of successive grants from NWO, Tilburg University and the Nijmegen–Tilburg CLS research school, and SoBU (the Cooperation Association of Brabant Universities). Its current crew consists of six Ph.D. students (three are shared with Eindhoven Technical University, and two of them are stationed in Eindhoven), one postdoc and one part-time guest postdoc, two part-time scientific programmers, and two senior researchers. The two major projects ILK is currently involved in, are an NWO Vernieuwingsimpuls *project entitled "Memory models of language" (2001-2005), and a Flemish-Dutch NWO-VNC project entitled "PROSIT: Prosody from Information in Text" (2001-2004), shared with the Universitaire Instelling Antwerpen (Antwerp University). More information and a selection of web-demos of machine-learned natural language processing systems can be found on the group's web site:* `http://ilk.kub.nl` .

1 Analogy and language

The heart of any natural language processing (NLP) system, be it for pronunciation, parsing, translation, or information extraction, is a computational mapping from an input symbol sequence to an output symbol sequence: from text to parsed text, from Dutch text to English text, or from a written sentence to its pronounced version. This mapping could involve one or several intermediate representations, when it "makes sense" (according to some optimization principle) to decompose the overall task into smaller ones. The model could be hand-built – but it need not be; when annotated data is available, it could also be learned automatically by machine learning methods. In fact, present-day research offers many demonstrations that NLP models can be learned pretty successfully with machine learning – and as I argue here, this is due to a mutual inherent basis, *analogy*. Machine learning offers productive models of analogy; language processing can largely be explained through analogy.

Throughout the development of linguistics in the twentieth century, several allusions have been made to this method of testing the efficacy of representation and modularisation through unbiased, simple methods that generalise by analogy (De Saussure, 1916). In the late 1980s, with the advent of desktop computers capable of tackling realistic problems, the first computational versions of models of analogy for language processing started to emerge. Partly, this movement came from linguistics (Skousen, 1989), but also from machine learning, in which researchers discovered

language processing data to be excellent test material for their algorithmic development (Sejnowski & Rosenberg, 1987; Dietterich & Bakiri, 1991).

2 Memory-based natural language processing

One classic formulation of analogy is the expression (A:B)::(A':B'). To read this expression from a language perspective, A and B denote symbol sequences at the same representation level that stand in a certain similarity relation. For example, they each denote a word (a sequence of letters). A' and B' are their respective corresponding symbol sequences at another representation level. For example, A' is the pronunciation of A, represented as a sequence of phonemes. The analogy expression thus states that "A is as similar to B as A' is to B'". Across different levels of language, this principle is essentially true, *provided that the right context is taken into account when similarity is estimated.*

This is a powerful statement, which opens the door to a large toolshed provided by a half century of artificial intelligence research, on systems that can use analogy productively to process new examples of a learned task. This may be illustrated by a simple incarnation of such a system that has two components. First, it has a memory capable of storing examples of mappings. For example, it has stored the fact that the word A corresponds to the phonemic representation A'. Second, it has an on-board similarity function that can estimate the similarity between words. When presented with the word B as a query, the system can go via the known correspondence relation between A and A' to what it believes must be B'; namely, that B' for which (A:B)::(A':B').

A slightly expanded variant of this system would be capable of searching for more than just the single-most similar example in memory: say it always searches for a fixed number k of the most closely similar memory examples. More similar cases could provide better support for estimating (interpolating, abducing) a correct B'. This is essentially what the k-nearest neighbor classifier (Fix & Hodges, 1951; Cover & Hart, 1967) does. This simple algorithm was developed in the field of pattern recognition and was reincarnated later in machine learning as instance-based learning, exemplar-based generalisation, and case-based reasoning (Stanfill & Waltz, 1986; Aha et al., 1991; Kolodner, 1993). To support our own natural language research, we developed the research-domain software package TiMBL (Tilburg Memory-Based Learner (Daelemans et al., 2001)). TiMBL implements the k-NN classifier with several variants of automatic similarity functions for symbolic sequences – also for non-linguistic symbol sequences or non-sequential symbolic task representations. From here on, I refer to the k-NN classifier as *memory-based learning.*

As an example application of this learner–classifier, consider the following language problem. In the sentence I ate pizza with salami, with salami is semantically attached to pizza. In the sentence I ate pizza with a fork, however, with a fork attaches to ate as being the instrument of eating. To understand these sentences (e.g. in order to answer questions about them), the correct attachment needs to be determined. Suppose now that a memory-based learner has stored many of these attachment cases, where each case has been annotated (by a human expert) as an attachment to the noun (with salami to pizza) or to the verb (with a fork to eat). Thus, the learner has memorised correspondences between sentences (sequences of words at particular slots) and an attachment symbol. We trained such a memory-based learner on English (Zavrel et al., 1997). When presented with the example unseen sentence The national oil company shifted the emphasis to gas, reduced to shift emphasis to gas to focus on just the necessary word information, this simple learner found the following six most (and equally) similar nearest neighbours to that string:

1. shift sales to quarter – verb attachment
2. shift money to ones – verb attachment
3. shift burden to prosecutor – verb attachment
4. shift blame to congress – verb attachment

5. shift weight to side – verb attachment
6. shift focus to fundamentals – verb attachment

All six are similar to shift emphasis to gas in the sense that they also contain shift and to. In all cases, the prepositional phrase to ... attaches to the verb. Interpolation of this unison mapping produces the estimation that in shift emphasis to gas, to gas also attaches to the verb shift – which according to human judgement would be correct.

Note that in this process, no explicit linguistic knowledge is manipulated. The linguistic analysis would be that the verb shift has a subcategorization frame with two complements; one noun phrase and one prepositional phrase starting with to. The memory-based classifier does not utilize any of these high-level constructs explicitly; rather, it retrieves the right amount of evidence from memory that contains the same information implicitly.

To elaborate on the last example, consider a set of alternative sentences with ... emphasis on ... in them. Usually, this construction means that the prepositional phrase starting with on is attached to the noun *emphasis*; like "shift to", "emphasis on" is a fixed phrase in English. Note then that similarity among symbol sequences is a step function when it is implemented as just counting overlapping words in the same positions: when place emphasis on gas would be in memory in the first example, then shift emphasis to gas would match this instance as much (viz. on two words) as it matches the six shift-to-examples above. Such ties can easily occur in symbol sequence matching when all symbols are regarded equally important in terms of matching score. To partly counteract that, and to incorporate more information about differences among input features in general, a typical memory-based classifier for language processing is equipped with *feature weighting* that assigns relatively high weights to relatively important features.

To phrase it slightly more technically, the standard similarity function with feature weighting (Daelemans & Van den Bosch, 1992) within the default memory-based classifier implemented in TiMBL is the *overlap metric* $\Delta(X,Y) = \sum_{i=1}^{n} w_i \; \delta(x_i, y_i)$, where $\Delta(X,Y)$ is the distance between patterns X and Y, represented by n features, w_i is the weight of feature i (see below) and δ is the distance of two values on the same feature: $\delta(x_i, y_i) = \; 0 \; if \; x_i = y_i; \; 1 \; if \; x_i \neq y_i$. The distance between two patterns is simply the sum of the differences between the features.

An established metric for estimating a sensible weight w_i of a symbolic feature is *information gain* (IG) weighting (Shannon, 1948), which looks at each feature in isolation, and measures how much information it contributes to our knowledge of the correct class label. The IG of feature i is measured by computing the difference in uncertainty (i.e. entropy) between the situations without and with knowledge of the value of that feature: $w_i = H(C) - \sum_{v \in V_i} P(v) \times H(C|v)$, where C is the set of class labels, V_i is the set of values for feature i, and $H(C) = -\sum_{c \in C} P(c) \log_2 P(c)$ is the entropy of the class labels. The probabilities are estimated from relative frequencies in the training set. An often-used adaptation of Information Gain, which is negatively sensitive to features with high numbers of values, is *gain ratio* (GR), which is IG divided by *split info* $si(i)$, the entropy of the feature-values (Quinlan, 1993): $w_i = \frac{H(C) - \sum_{v \in V_i} P(v) \times H(C|v)}{si(i)}$ where $si(i) = -\sum_{v \in V_i} P(v) \log_2 P(v)$.

The resulting Gain Ratio values can then be used as weights w_f in the weighted distance metric. In the example used earlier, the identity of the preposition (on, to) has a gain ratio of about 0.9, while the three other word slots have gain ratios of about 0.3. Given these values (determined on all memorised material, thus "global" information), shift emphasis to gas would be more similar –and rightly so, in this case– to any of the six shift to examples given earlier than to place emphasis on gas, because the match on both shift and to is stronger than the match on both emphasis and gas.

In practice, memory-based learning, which in its TiMBL incarnation is basically the above algorithm, equipped with a number of additional and alternative metrics for similarity, feature

weighting, and class voting (Daelemans et al., 2001), has been applied to a range of realistic and complex natural language tasks with considerable success. The applications range from morphological and phonological processing (pronunciation, hyphenation, morphological analysis) to full text pronunciation, parsing, word sense disambiguation, question answering and aspects of man–machine dialogue management[1]. To cut this applied story short, it turns out that analogy can indeed be used to process natural language, and that actual NLP systems are produced along the way that have serious commercial applicability.

3 Task representation

More seriously, the crossing of machine learning with NLP has also produced a number of empirically-based insights that are of importance to linguistics. The experimenter who applies machine learning methods to language data is essentially a linguist – the machine learning paradigm offers a benchmarking environment for testing linguistic hypotheses. Which feature representation (including feature selection and construction, and task decomposition) makes a learnable language model that generalises well to new data?

Here is an example of such an application study and the "lesson learned". The example concerns a partial form of parsing that resembles the type of parsing taught at high school. Consider the sentence At the prearranged time, Greg started the engine and taxied out. A partially parsed version of this sentence reads:

> [[At]$_{PP-temporal}$ [the prearranged time]$_{NP}$]$_{PNP-temporal}$, [Greg]$_{NP-subject}$ [started]$_{VP/S}$ [the engine]$_{NP-object}$ and [taxied]$_{VP/S}$ [out]$_{ADV-direction}$.

This level of parsing is useful for information extraction and question answering. When the question would be, "when did Greg start the engine?", then parsing this sentence in the above way would produce "at the prearranged time" as the exact answer, since that chunk is labeled as a temporal prepositional noun phrase (PNP) of the sentence in which "Greg" is the subject, "started" one of the main verbs, and "the engine" object.

A relatively undisputed standard assumption in computational linguistics is that the input for this task (a sentence composed of words) needs to be either transformed or enriched by another level of linguistic representation, namely part-of-speech (POS) tags, in order to be good input for the final partial-parsing step. POS tags are common placeholder terms for groups of words that behave syntactically the same (or similarly): nouns, verbs, pronouns, prepositions, adjectives, adverbs, determiners, interjections, etc. Using POS tags *instead* of words alleviates the problem of sparseness: many words occur only a few times, and therefore form a weak basis for feature weighting or matching. POS tags, in contrast, come in small numbers (usually between 10 and 300, depending on the language and the exact task definition) and consequently high frequencies, so relatively small quantities of training material (e.g. a few thousand sentences) suffice to give the machine learner enough examples of the larger part of possible different sequences. Moreover, POS tags, once correctly determined, disambiguate the syntactic function of words such as suspect, which may be a noun or verb; for a task such as partial parsing this is relevant information.

For English, a large annotated of parsed text exists. It should be noted that this is the result of a very costly effort – it involves human annotation, which is hard and time-consuming. The corpus consists of Wall Street Journal articles and a mixed bag of other texts, and totals about 74k sentences.

One experiment we performed was to learn the above partial parsing task with (i) an input consisting only of words, and (ii) and input consisting only of the hand-annotated POS tags.

[1] See the ILK home page for publications on these topics: http://ilk.kub.nl .

# training sentences	word input	POS-tag input
100	39.6	61.3
500	48.7	66.6
1000	52.4	68.1
5000	62.2	70.4
10000	66.4	71.6
50000	74.2	73.5
74029	75.4	73.9

Table 1: F-scores on test material (10-fold cross validation averages) on increasing numbers of training sentences, with words and POS tags as input.

The latter input is arguably the most correct POS tagging available; in any system which has to determine POS tags by itself, POS accuracy would be between 95% and 98%. Both inputs were constrained to fixed "windows" of seven words or tags, and the target mapping was the partial parse code on the middle word of the window (a partial parse can be encoded word-by-word since it is an almost flat encoding). Table 1 displays the F-score, a harmonic mean of precision (how many chunks predicted by the learner are correctly parsed) and recall (how many chunks in the test material were correctly parsed by the learner), using both inputs on different sizes of the learning material, with 10% of all material as test set (repeated 10 times in a 10-fold cross validation experiment). The learner is a memory-based learner with feature weighting, a k set to 7, distance-weighted class voting, and a modified value-difference metric (Daelemans et al., 2001).

Somewhere between 10k and 50k sentences of training material, an input of words produces better partial parsing of unseen material than an input of POS tags. Apparently in that area the sparseness problem of low-frequent words is solved to a degree that the lexical information in words begins to outweigh the flattened, non-lexical information in POS tags. Recalling the earlier example of attaching prepositional phrases, word input provides the difference betweeen to and on, whereas a standard POS tagging would consider both simply as prepositions, thereby throwing away possibly discriminatory information. As for the disambiguation information in POS tags (the suspect example), it is easy to imagine that words provide the same information implicitly, given enough examples; word contexts such as the suspect and we suspect disambiguate the two syntactic functions as well, so when either of these contexts appears in new material, they will be properly matched and coupled to the right parsing information – again, given that these examples have been stored in memory. In all, the results presented here suggest that that the loss of information by replacing words by POS tags leads to an eventual slack in precision and recall.

As seen from Table 1, the score on word features appears to climb with a near-constant rate when the training set becomes ten times larger. This has been witnessed in other large-scale experiments as well (Banko & Brill, 2001), and can be explained by the fact that words are not distributed normally, but Zipfian (Zipf, 1935): in short, there is a small group of words that occurs very frequently (mostly function words: determiners, prepositions, conjunctions) and a large group of words that occur infrequently (mostly content words: nouns, verbs, adjectives). It is not unusual that half of the words in a large text collection occur only once in that collection; doubling the size of the collection basically adds just as much new unseen words, while improving the frequency of observing slightly more frequent words. Because of this inherent language feature, the information provided by more data simply keeps improving the information implicit in that data, even with very very large datasets of 1G (1 billion) words (Banko & Brill, 2001).

4 Discussion

The latter example study is typical of the information that the application of machine learning to NLP tasks produces. This particular example, and also my thesis (Van den Bosch, 1997) for the case of English word pronunciation, shows that intermediate levels proposed or assumed necessary by traditional (computational) linguistic expert models, are worse sources of information than low-level, zero-cost surface features such as words. This can be brought back to an issue of explicitness versus implicitness of information: expert-based models tend to overestimate the importance of making hidden representations (such as POS tags) explicit, while at the same time they underestimate the information that is implicitly present in surface features. This contrast bears a strong relation with the old "computation versus memory" discussion in (computational) linguistics; our results suggest that memory can be relied on much more intensely due to the implicit information that can be mined from it easily, through simple analogy-based models.

In the ongoing ILK group projects, particularly in the Vernieuwingsimpuls project "Memory models of language", we are deepening this type of study on a broad range of NLP tasks. We hope to open up new ways of reporting back to linguistics that has become available through the interplay of machine learning, the availability of annotated data, and the advance in typical computer speed and memory.

References

Aha, D. W., Kibler, D., & Albert, M. (1991). Instance-based learning algorithms. *Machine Learning, 6*, 37–66.

Banko, M., & Brill, E. (2001). Scaling to very very large corpora for natural language disambiguation. *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL'01)*.

Cover, T. M., & Hart, P. E. (1967). Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory, 13*, 21–27.

Daelemans, W., & Van den Bosch, A. (1992). Generalisation performance of backpropagation learning on a syllabification task. *Proc. of TWLT3: Connectionism and Natural Language Processing* (pp. 27–37). Enschede.

Daelemans, W., Zavrel, J., Van der Sloot, K., & Van den Bosch, A. (2001). *TiMBL: Tilburg Memory Based Learner, version 4.0, reference manual* (Technical Report ILK-0104). ILK, Tilburg University.

De Saussure, F. (1916). *Course de linguistique générale.* Paris: Payot. edited posthumously by C. Bally and A. Riedlinger.

Dietterich, T. G., & Bakiri, G. (1991). Error-correcting output codes: A general method for improving multiclass inductive learning programs. *Proceedings of AAAI-91* (pp. 572–577). Menlo Park, CA.

Fix, E., & Hodges, J. L. (1951). *Disciminatory analysis—nonparametric discrimination; consist ency properties*Technical Report Project 21-49-004, Report No. 4). USAF School of Aviation Medicine.

Kolodner, J. (1993). *Case-based reasoning.* San Mateo, CA: Morgan Kaufmann.

Quinlan, J. (1993). C4.5*: Programs for machine learning.* San Mateo, CA: Morgan Kaufmann.

Sejnowski, T. J., & Rosenberg, C. S. (1987). Parallel networks that learn to pronounce English text. *Complex Systems, 1*, 145–168.

Shannon, C. (1948). A mathematical theory of communication. *Bell System Technical Journal, 27*, 379–423 and 623–656.

Skousen, R. (1989). *Analogical modeling of language.* Dordrecht: Kluwer Academic Publishers.

Stanfill, C., & Waltz, D. (1986). Toward memory-based reasoning. *Communications of the* ACM, *29*, 1213–1228.

Van den Bosch, A. (1997). *Learning to pronounce written words: A study in inductive language learning.* Doctoral dissertation, Universiteit Maastricht.

Zavrel, J., Daelemans, W., & Veenstra, J. (1997). Resolving PP attachment ambiguities with memory-based learning. *Proc. of the Workshop on Computational Language Learning (CoNLL'97).* ACL, Madrid.

Zipf, G. K. (1935). *The psycho-biology of language: an introduction to dynamic philology.* Cambridge, MA: The MIT Press. Second paperback edition, 1968.

6.3 Fingerprints from Quantum Mechanics: Ronald de Wolf

rdewolf@cs.berkeley.edu

Introduction

Two dead bodies have been found on the same night in different parts of town, strangled to death. The police are anxious to know whether the killer in both cases was the same person. Fortunately, the murderers were stupid enough to leave their fingerprints on the necks of the victims, so the police only need to compare fingerprints from both necks to see whether they match or not. This will tell them whether both murderers are actually the same person.

This is an interesting phenomenon. The reason it works is that every human being has his own unique fingerprint. Such fingerprints do not give much information about their respective owners (assuming we do not have a complete table matching all possible fingerprints with all possible humans), but they *do* allow us to test for identity: if we want to know whether John and Jack are the same person, it suffices to compare their fingerprints — no need for John or Jack to be present themselves! However, human fingerprints are clearly restricted to human beings, and most other objects (houses, computers, cans of tomatoes) do not have fingerprints in a similar way. Here we will describe a general method that allows us to take short fingerprints of anything that can be described in bits. That is, we will associate with each n-bit string an exponentially smaller fingerprint, such that identity between two strings can be detected by comparing their fingerprints. The caveat is that our fingerprints will need to be *quantum mechanical*: they will be *superpositions* of classical states. This quantum fingerprinting method allows us to do certain things that are provably impossible in the world of classical physics and classical computing.

This article describes joint work with Harry Buhrman (CWI and UvA), Richard Cleve (Calgary), and John Watrous (Calgary), published recently in [3]. Before describing our quantum fingerprinting scheme, we will first give a brief introduction to quantum states and their use in computation.

Quantum computing

States and operations

In a classical computer, the unit of information is a *bit*, which can take on the values 0 or 1. In a quantum computer, the unit is a *quantum* bit, which is a linear combination of those two values. That is, a qubit is a superposition of the two "basis states" $|0\rangle$ and $|1\rangle$:

$$\alpha_0|0\rangle + \alpha_1|1\rangle,$$

where complex number α_0 is called the *amplitude* of the basis state $|0\rangle$, and α_1 is the amplitude of $|1\rangle$. We require $|\alpha_0|^2 + |\alpha_1|^2 = 1$. Viewing $|0\rangle$ and $|1\rangle$ as the vectors $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ respectively, the qubit corresponds to $\begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}$. In a way, such a qubit is in both classical states simultaneously.

More generally, the state $|\phi\rangle$ of an m-qubit quantum computer can be described by a superposition of all 2^m classical m-bit states:

$$|\phi\rangle = \sum_{i\in\{0,1\}^m} \alpha_i|i\rangle,$$

with the condition that the squared amplitudes sum to 1: $\sum_i |\alpha_i|^2 = 1$. We can also view this state as the 2^m-dimensional complex unit vector that has the α_i as amplitudes.

There are basically two ways in which a quantum computer can manipulate such a state: it can make a measurement or apply a unitary transformation. Suppose we measure state $|\phi\rangle$. We cannot "see" a superposition itself, but only classical states. Accordingly, if we measure $|\phi\rangle$ we will see one and only one classical m-bit state $|i\rangle$. Which specific $|i\rangle$ will we see? This is not determined in advance; the only thing we can say is that we will see state $|i\rangle$ with probability $|\alpha_i|^2$. Because $|\phi\rangle$ is a unit vector, these probabilities nicely sum to 1. If we measure $|\phi\rangle$ and see classical state $|i\rangle$ as a result, then $|\phi\rangle$ itself has "disappeared", and all that is left is $|i\rangle$. In other words, observing $|\phi\rangle$ "collapses" the quantum superposition $|\phi\rangle$ to the classical state $|i\rangle$ that we saw, and all information that might have been contained in the other amplitudes is gone.

Instead of measuring $|\phi\rangle$, we can also apply some operation to it, i.e., change the state to some

$$|\psi\rangle = \sum_{i \in \{0,1\}^m} \beta_i |i\rangle.$$

Quantum mechanics only allows *linear* operations to be applied to quantum states, so the operation must correspond to multiplying the vector $|\phi\rangle$ with some matrix U:

$$U \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{2^m-1} \end{pmatrix} = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_{2^m-1} \end{pmatrix}.$$

Because $|\psi\rangle$ should also be a unit vector, we have the constraint that U preserves norm, and hence is unitary (that is, U^{-1} equals the conjugate transpose U^*). Just like Boolean circuits, a well-chosen unitary matrix U followed by an appropriate measurement can compute any computable function. Every U can be built up from a small number of "elementary gates". These are unitary transformations that each act on only one or two qubits, just as classical Boolean AND, OR, and NOT gates act on only one or two bits. A quantum computation is deemed efficient to the extent that U can be built up from a small number of such elementary gates.

As a simple example, consider the 1-qubit *Hadamard* gate, specified by the following unitary matrix:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

If we apply this to the classical state $|0\rangle$, we obtain the equal superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. If we measure this, we will see either $|0\rangle$ or $|1\rangle$, each with probability 50%. If we apply H to $|1\rangle$, then we obtain $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, which induces the same probability distribution when measured. However, if we apply H to a superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, then we get the classical state $|0\rangle$ back, because the positive and negative contributions to the amplitude of $|1\rangle$ add up to 0. This effect is known as *interference*.

What is it good for?

Why should we consider quantum computing? On a fundamental level, the answer is that computers are physical systems and physical systems are quantum mechanical. Accordingly, if we want to study the ultimate power and limits of computers, we should consider the full power and limits of quantum mechanics.

On a moderately more practical level, the main reason to consider quantum computers is that they can solve certain computational problems much faster than classical computers. For most computational problems, a quantum computer is not significantly more efficient than a classical computer (most problems are hard by any standard — classical as well as quantum), but for some it is.

The most important example of this is the problem of finding prime factors of large numbers. Peter Shor's quantum algorithm from 1994 [8] finds a factor of an n-bit number in roughly n^2

steps (elementary gates). In contrast, the best classical algorithms that we know, need about $2^{n^{1/3}}$ steps to find a factor. Even with massive parallelism, todays computers need several months to factor 512-bit numbers — and rightly so, because much of modern cryptography would become completely insecure if computers could quickly factor numbers of 512 or 1024 bits. In principle quantum computers could do this, but practice lags far behind theory in this young field. The largest number factored by a quantum computer to date is 15(=3*5), on a 7-qubit quantum computer [9].

A second example where quantum computers are much faster than classical ones is the problem of searching an unordered set of N elements for some target element. For example, searching for the person with phone number 5260248 in a phone directory that is ordered by name but not by phone number. Grover's quantum search algorithm from 1996 [5] finds the target element in about $\sqrt{N}$ steps, while a classical algorithm can do no better than just go through all records sequentially, which takes N steps. For example, Grover's algorithm can find a satisfying assignment for an n-bit Boolean formula in roughly $\sqrt{2^n}$ steps, while classical exhaustive search would have to go over all 2^n possible truth assignments separately.

How to construct short quantum fingerprints and test them

We will now use quantum states to construct a fingerprinting scheme. Recall the main idea behind fingerprinting: we want to map large objects (n-bit strings) to short objects (their fingerprints), such that we can decide whether two such large objects are equal by comparing only their fingerprints. A good quantum fingerprinting scheme thus requires two things: (1) a mapping from n-bit strings x to their short quantum fingerprints $|\phi_x\rangle$ and (2) a test to decide whether $x = y$, given only fingerprints $|\phi_x\rangle$ and $|\phi_y\rangle$.

It is not hard to show that non-orthogonal states (= states with non-zero inner product) cannot be distinguished with probability 1. Thus, if we want our test to work perfectly, the fingerprints $|\phi_x\rangle$ and $|\phi_y\rangle$ would need to be exactly orthogonal for all pairs of distinct n-bit strings x and y. Unfortunately, this constraint makes the quantum fingerprints way too long: an orthonormal set of 2^n states requires 2^n dimensions, which corresponds to n qubits — not much savings over n classical bits! Instead we will settle for *near*-orthogonality, where the required number of dimensions can be made much smaller. Giving up exact orthogonality implies that our test will have a certain error probability, but we can make this error probability as small as we want.

There are many ways to obtain a set of 2^n near-orthogonal states in a small number of dimensions. Below we will use a simple application of the probabilistic method for this, but more constructive methods based on sophisticated error-correcting codes exist as well.

Suppose we pick a set S of 2^n d-bit strings at random, for some d to be determined later. Then the expected Hamming distance $H(s,t)$ between two such strings s and t is $d/2$, and the Chernoff bound tells us that the actual distance is probably close to its expectation:

$$\Pr\left(H(s,t) \notin [0.49d, 0.51d]\right) \leq 2^{-cd}$$

for some positive constant c. Now suppose we choose $d = 2n/c$, then the above probability is at most 2^{-2n} and using the union bound we have

$$\begin{aligned}\Pr\left(\exists s,t \in S \text{ with } H(s,t) \notin [0.49d, 0.51d]\right) &\leq \sum_{s,t\in S} \Pr\left(H(s,t) \notin [0.49d, 0.51d]\right) \\ &\leq \binom{2^n}{2} 2^{-2n} < 1.\end{aligned}$$

In particular, there exists at least one set S where $H(s,t) \in [0.49d, 0.51d]$ *for all* distinct $s,t \in S$. Let us consider such a set. We can index the elements in $S = \{s^x \mid x \in \{0,1\}^n\}$ by the n-bit strings, and derive quantum states from them by using the bits s_i^x in s^x for signs of amplitudes in

$|\phi_x\rangle$:

$$|\phi_x\rangle = \frac{1}{\sqrt{d}} \sum_{i=1}^{d} (-1)^{s_i^x} |i\rangle.$$

Since these states live in $d = 2n/c$ dimensions, we only need $\log d = \log n + O(1)$ qubits to represent them, so our quantum fingerprints are indeed short compared to the underlying n-bit strings. Two fingerprints are almost orthogonal, because the inner product between $|\phi_x\rangle$ and $|\phi_y\rangle$ is

$$\frac{1}{d} \sum_{i=1}^{d} (-1)^{s_i^x + s_i^y}.$$

Because the Hamming distance between s^x and s^y is close to $d/2$, $s_i^x + s_i^y$ will be even for about half of the is and odd for the other half. Therefore the above sum contains about as many $+1$s as -1s and hence will be small.

We now have our mapping from strings to short quantum fingerprints. It remains to show how we can test whether $x = y$, when given only fingerprints $|\phi_x\rangle$ and $|\phi_y\rangle$. Our test is pictured in Figure 1, where time progresses from left to right: we add on an auxiliary $|0\rangle$-qubit, apply a Hadamard transform to it to get $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, apply a controlled swap to the two registers containing the fingerprints (this swaps the two registers if the auxiliary qubit is $|1\rangle$ and does nothing if it is $|0\rangle$), then apply another Hadamard transform, and finally measure the auxiliary qubit.

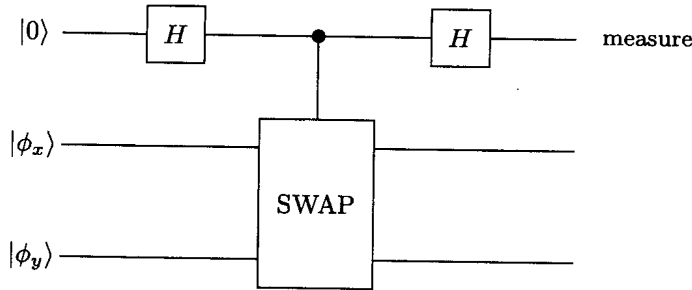

Figure 1: Test whether 2 fingerprints $|\phi_x\rangle$ and $|\phi_y\rangle$ are equal

Let us analyze what happens here. First, if $x = y$ then $|\phi_x\rangle = |\phi_x\rangle$ and the controlled swap has no effect, since swapping two identical things doesn't do anything. In this case, the second Hadamard transform will just set the auxiliary qubit back to the $|0\rangle$-state, so our measurement will give outcome 0 with certainty. On the other hand, if $x \neq y$ then $|\phi_x\rangle$ and $|\phi_y\rangle$ are almost orthogonal. In this case, by calculating the final state one can show that the measurement will give outcome $|1\rangle$ with probability close to $1/2$ (where the "closeness" depends on the inner product between $|\phi_x\rangle$ and $|\phi_y\rangle$). Thus one such test allows us to distinguish the two cases $x = y$ and $x \neq y$ with one-sided error probability about $1/2$. If we have a few copies of both fingerprints available, then we can repeat the above test and reduce the error probability to a small constant.

Application: Saving communication

We now describe an application of our quantum fingerprinting scheme. We will consider a simple communication scenario. There are three parties: Alice, Bob, and a referee. Alice receives n-bit input x and Bob receives n-bit input y. The referee receives no input, but he wants to find out whether $x = y$ or not (the *equality* problem). Alice and Bob each can send information to the referee, but cannot receive messages from the referee, nor can they communicate with each other. We want a scheme that uses only little communication, but that allows the referee to determine whether $x = y$ with high probability, for all inputs x, y.

Clearly, Alice can send the whole x and Bob can send the whole y, allowing the referee to solve the problem at a cost of $2n$ bits of communication. However, smarter things with less communication are possible. The classical communication complexity of this problem has been studied by various researchers in the last decade, and it turns out that about $\sqrt{n}$ bits of communication are sufficient [1] as well as necessary [6, 2] to solve this equality problem.[1] In contrast, the construction of short quantum fingerprints together with the equality test outlined above, immediately suggest a much more efficient *quantum* solution to the equality problem: Alice sends the fingerprint $|\phi_x\rangle$ to the referee (or a few copies thereof), Bob sends the fingerprint $|\phi_y\rangle$, and the referee just tests whether the two fingerprints he received are equal or almost orthogonal. This gives us a solution to the equality problem that works with high success probability and requires only $O(\log n)$ qubits to be sent, which is exponentially better than the $\sqrt{n}$ bits of communication that are required classically (this also implies that there is no efficient *classical* fingerprinting scheme that achieves the same as our quantum scheme).

For example, suppose Alice and Bob are flying through space, each in their own spaceship. They can only send messages to the command center on earth. They have each gathered a large chunk of data, of 2^{40} bits say, and for some reason the command center needs to know whether they have *the same* chunk of data. Classically, Alice and Bob would each need to send about $\sqrt{2^{40}} \approx 1,000,000$ bits to the referee. In the quantum case, only about 50 qubits of communication would already suffice — a significant savings.

Conclusion

We described the quantum fingerprinting technique from [3]. To each n-bit string x we can associate a $\log n$-qubit state $|\phi_x\rangle$, such that we can decide whether $x = y$ by deciding whether $|\phi_x\rangle = |\phi_y\rangle$. In other words, for the purposes of identification the long object x can be replaced by its short fingerprint $|\phi_x\rangle$. This gives rise to an exponential reduction in the communication complexity of the equality problem when we allow quantum communication.

What about other applications of quantum fingerprinting? Note that the fingerprint $|\phi_x\rangle$ gives only little information about x, because a $\log n$-qubit state can contain only $\log n$ bits of classical information (Holevo's theorem). In some sense the quantum fingerprint "contains" x completely without revealing it. Yet we can clearly test or verify whether the hidden x equals some string y of our choice, by testing $|\phi_x\rangle$ against $|\phi_y\rangle$. This information-hiding property of quantum fingerprints smacks of cryptography, and indeed there has recently been some work on "quantum signatures" that uses quantum fingerprints as a building block [4]. Further applications of quantum fingerprinting in communication complexity or cryptography may lie ahead.

References

[1] A. Ambainis. Communication complexity in a 3-computer model. *Algorithmica*, 16(3):298–301, 1996.

[2] L. Babai and P. G. Kimmel. Randomized simultaneous messages: Solution of a problem of Yao in communication complexity. In *Proceedings of the 12th Annual IEEE Conference on Computational Complexity*, pages 239–246, 1997.

[3] H. Buhrman, R. Cleve, J. Watrous, and R. de Wolf. Quantum fingerprinting. *Physical Review Letters*, 87(16), September 26, 2001. http://xxx.lanl.gov/abs/quant-ph/0102001.

[4] D. Gottesman and I. Chuang. Quantum signatures. quant-ph/0105032, 8 May 2001.

[5] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th ACM STOC*, pages 212–219, 1996. quant-ph/9605043.

[1] Only $O(1)$ classical bits of communication would suffice if Alice and Bob had access to some shared source of randomness, but we're not allowing that here.

[6] I. Newman and M. Szegedy. Public vs. private coin flips in one round communication games. In *Proceedings of 28th ACM STOC*, pages 561–570, 1996.

[7] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[8] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. Earlier version in FOCS'94. quant-ph/9508027.

[9] L. Vandersypen, M. Steffen, G. Breyta, C. Yannoni, M. Sherwood, and I. Chuang. Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414(25):883–887, 2001. quant-ph/0112176.