

COMMUNICATIONS

CACM.ACM.ORG

OF THE

ACM

09/2010 VOL.53 NO.09

Computers in Patient Care

Why has progress been so slow?

Performance Evaluation
and Model Checking
Join Forces

Point/Counterpoint
on Future Internet
Architecture

Erlang

Degrees, Distance,
Dollars

3RD ACM SIGCHI
SYMPOSIUM ON

ENGINEERING INTERACTIVE COMPUTING SYSTEMS

NOVEMBER 22, 2010

Submission deadline
for Long Papers and Workshops

FEBRUARY 10, 2011

Submission deadline
for Late Breaking Results,
Demos, Doctoral Consortium,
Tutorials

LOCATION

Area della Ricerca CNR
Pisa, ITALY

CONFERENCE CHAIR

Fabio Paternò, CNR-ISTI, HIIS
Laboratory

LONG PAPER CHAIRS

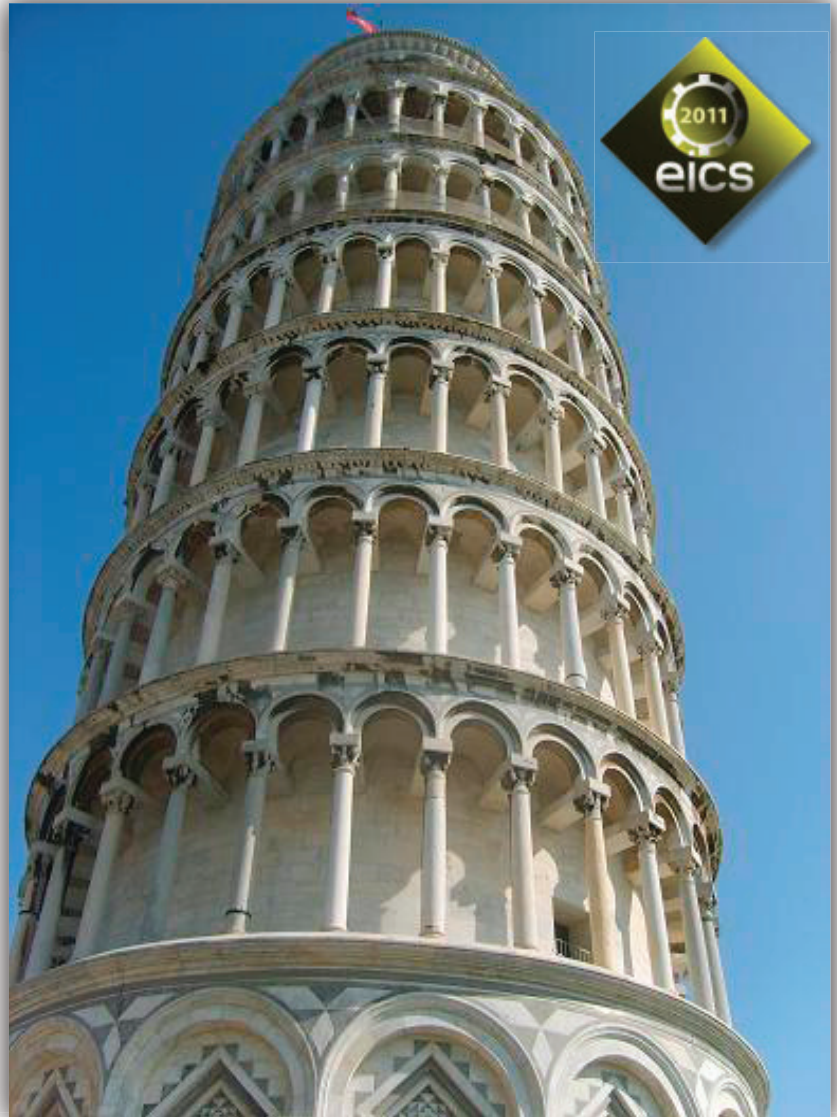
Kris Luyten, Hasselt University
Frank Maurer, Univ. of Calgary

Late Breaking Results CHAIRS

Prasun Dewan, UNC Chapel Hill
Carmen Santoro, CNR-ISTI, HIIS
Laboratory

For information, please contact
info@eics2011.org

EICS is sponsored by
ACM SIGCHI



EICS 2011

PISA, Italy

JUNE 13-16, 2011

Topics include but are not limited to:

- Integrating interaction design into the software development process,
- Engineering processes for interactive systems (e.g. design, implementation, prototyping and testing),
- Dynamic generation/composition of interactive systems,
- Requirements engineering for interactive systems,
- Software architectures for interactive systems,
- Modeling interaction and interactive systems,
- Innovative Interactive Applications,
- Interactive systems specification,
- Specifying users' activities,
- Designing Usable Software.

More updated information at
WWW.EICS2011.ORG



Search Computing Challenges and Directions

S. Ceri, M. Brambilla,
Politecnico di Milano,
Italy (Eds.)

This clearly structured book contains detailed papers on the subject of search computing. It focuses on building the answers to complex queries on the Web. The papers, written by leading scientists, contain the latest results in the field.

2010. X, 321 p. (Lecture Notes in Computer Science / Information Systems and Applications, incl. Internet/Web, and HCI, Volume 5950) Softcover
ISBN 978-3-642-12309-2 ► **\$83.00**



Agile Software Development Current Research and Future Directions

T. Dingsøyr, T. Dybå,
N. B. Moe, SINTEF ICT,
Trondheim, Norway (Eds.)

Addresses both sources of the agile confusion: fuzzy, multifaceted scope and poor, unconsolidated dissemination of research results and experiences. Based on many years of academic research and industrial experience. Contributions written by leading experts in agile software development.

2010. XVII, 240 p. Hardcover
ISBN 978-3-642-12574-4 ► **\$89.95**



OSS Design Patterns A Pattern Approach to the Design of Telecommunications Management Systems

C. Ashford, Ottawa,
Ontario, ON, Canada; P. Gauthier, Metasolv LLC
Offers guides to a standardization by a higher level of architectural agreement-design patterns. Provides an enduring architectural-level direction that will guide developers.

2009. XVI, 152 p. 31 illus. Hardcover
ISBN 978-3-642-01395-9 ► **\$79.95**



The Integrated Architecture Framework Explained

Why, What, How

J. van't Wout,
M. Waage, H. Hartman,
M. Stahlecker,

A. Hofman, Capgemini Nederland B.V., Utrecht,
The Netherlands

Only full authorized guide to the Integrated Architecture Framework. Written by its original developers. Based on many years of experience in enterprise architecture.

2010. 200 p. Hardcover
ISBN 978-3-642-11517-2 ► **\$59.95**



Handbook on Business Process Management 1

Introduction,
Methods and
Information Systems

J. vom Brocke, University of
Liechtenstein,

Vaduz, Principality of Liechtenstein;
M. Rosemann, Queensland University of Technology,
Brisbane, QLD, Australia (Eds.)

Gives a comprehensive understanding of Business Process Management. Covers the six main components of Business Process Management, i. e. strategic alignment, governance, methods, information technology, people and culture. Key features are the overall composition of the content based on a well-defined maturity model and the outstanding quality of the contributors.

2010. 600 p. (International Handbooks on Information Systems) Hardcover
ISBN 978-3-642-00415-5 ► **\$239.00**

Handbook on Business Process Management 2

Strategic Alignment, Governance,
People and Culture

2010. XVIII, 602 p. (International Handbooks on Information Systems) Hardcover
ISBN 978-3-642-01981-4 ► **\$239.00**

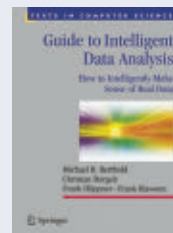


Cloud Computing Principles, Systems and Applications

N. Antonopoulos,
University of Derby, UK;
L. Gillam, University of
Surrey, UK (Eds.)

Includes a Preface by Professor Mark Baker of the University of Reading, UK. Presents the principles, techniques, protocols and algorithms that can be adapted from other distributed computing paradigms to the development of successful Clouds Elaborates the economic schemes needed for Clouds to become viable business models.

2010. XVI, 398 p. (Computer Communications and Networks) Hardcover
ISBN 978-1-84996-240-7 ► **\$129.00**



Guide to Intelligent Data Analysis

How to Intelligently
Make Sense of Real
Data

M. R. Berthold,
C. Borgelt, F. Höppner,

F. Klawonn,

Presents a broad-range of perspectives on data analysis, providing readers with a comprehensive account of the field. Focuses on the practical aspects as well as presenting the theory comprehensively. A special emphasis is given to put on pointing out the pitfalls that lead to wrong or insufficient analysis of results.

2010. XII, 397 p. (Texts in Computer Science, Volume 42) Hardcover
ISBN 978-1-84882-259-7 ► **approx. \$89.95**

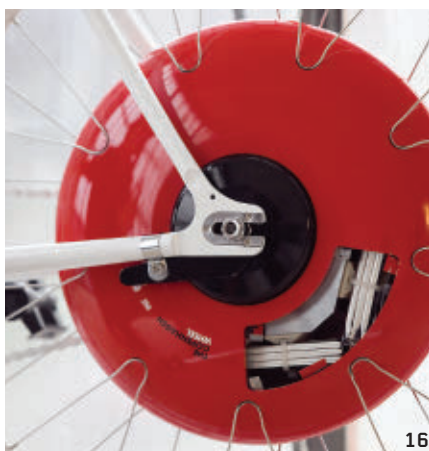
Departments

- 5 **Editor's Letter**
Science Has Only Two Legs
By Moshe Y. Vardi
-
- 7 **Letters To The Editor**
More Than One Way to Annotate Metadata
-
- 8 **BLOG@CACM**
Expanding CS Education; Improving Software Development
Ed H. Chi writes about the social Web's impact on CS education. Ruben Ortega discusses software and test-driven development.
-
- 12 **CACM Online**
More Communications
By David Roman
-
- 25 **Calendar**
-
- 107 **Careers**

Last Byte

- 110 **Puzzled**
Solutions and Sources
By Peter Winkler
-
- 112 **Future Tense**
Little Brother Is Watching
In a world of technology and fear, the public gets to know what it wants to know... and more than it can possibly digest.
By Greg Bear

News



- 13 **Brains and Bytes**
Computational neuroscientists are learning that the brain is like a computer, except when it isn't.
By David Lindley
-
- 16 **Cycling Through Data**
Sensor-equipped bicycles are providing valuable data to cyclists, city planners, and computer scientists.
By Neil Savage
-
- 18 **Degrees, Distance, and Dollars**
The Internet is making higher education accessible to a whole new class of students—but not necessarily at a lower cost.
By Marina Krakovsky
-
- 20 **ACM China Nearing Launch**
ACM's expansion into China will support local professionals and increase Chinese involvement in ACM's international activities.
By Tom Geller
-
- 21 **Kyoto Prize and Other CS Awards**
László Lovász, Vinton G. Cerf, and other researchers are honored for their contributions to computer science.
By Jack Rosenberger

Viewpoints

- 23 **The Business of Software Return at Risk**
Calculating the likely true cost of projects.
By Phillip G. Armour
-
- 26 **Law and Technology**
Principles of the Law of Software Contracts
An overview of a new set of legal principles for software contracts developed by the American Law Institute.
By Robert A. Hillman and Maureen A. O'Rourke
-
- 29 **The Profession of IT**
Discussing Cyber Attack
Cyber attack—the other side of cyber defense—deserves a more open discussion than it has been getting.
By Peter J. Denning and Dorothy E. Denning
-
- 32 **Viewpoint**
Objects Never? Well, Hardly Ever!
Revisiting the Great Objects Debate.
By Mordechai Ben-Ari
-
- 36 **Point/Counterpoint**
Future Internet Architecture: Clean-Slate Versus Evolutionary Research
Should researchers focus on designing new network architectures or improving the current Internet?
By Jennifer Rexford and Constantine Dovrolis

Practice



- 42 **Computers in Patient Care: The Promise and the Challenge**
Information technology has the potential to radically transform health care. Why has progress been so slow?
By Stephen V. Cantrill, M.D.

- 48 **Injecting Errors for Fun and Profit**
Error-detection and correction features are only as good as our ability to test them.
By Steve Chessin

- 55 **Thinking Clearly About Performance, Part 1**
Improving the performance of complex software is difficult, but understanding some fundamental principles can make it easier.
By Cary Millsap



Articles' development led by **acmqueue**
queue.acm.org

Contributed Articles



- 62 **Confronting the Myth of Rapid Obsolescence in Computing Research**
Computing research ages more slowly than research in other scientific disciplines, supporting the call for parity in funding.
By Dag I.K. Sjøberg

- 68 **Erlang**
The same component isolation that made it effective for large distributed telecom systems makes it effective for multicore CPUs and networked applications.
By Joe Armstrong



artist Paul Price; <http://www.paulprice.org.uk/>

About the Cover:

"With all of the computerization of so many aspects of our daily lives, medical informatics has had limited impact on day-to-day patient care," says Stephen V. Cantrill, M.D. in this month's cover story. Dr. Cantrill examines the many issues and challenges impeding faster progress in this field. Cover image by London-based award-winning graphic

Review Articles

- 76 **Performance Evaluation and Model Checking Join Forces**
A call for the perfect marriage between classical performance evaluation and state-of-the-art verification techniques.
*By Christel Baier,
Boudewijn R. Haverkort,
Holger Hermanns,
and Joost-Pieter Katoen*

Research Highlights

- 88 **Technical Perspective**
Programming with Differential Privacy
By Johannes Gehrke
- 89 **Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis**
By Frank McSherry
- 98 **Technical Perspective**
Constraint Satisfaction Problems and Computational Complexity
By Mark Jerrum
- 99 **Constraint Satisfaction Problems and Global Cardinality Constraints**
Andrei A. Bulatov and Dániel Marx



Association for Computing Machinery
Advancing Computing as a Science & Profession



ACM, the world's largest educational and scientific computing society, delivers resources that advance computing as a science and profession. ACM provides the computing field's premier Digital Library and serves its members and the computing profession with leading-edge publications, conferences, and career resources.

Executive Director and CEO

John White

Deputy Executive Director and COO

Patricia Ryan

Director, Office of Information Systems

Wayne Graves

Director, Office of Financial Services

Russell Harris

Director, Office of Membership

Lillian Israel

Director, Office of SIG Services

Donna Cappel

Director, Office of Publications

Bernard Rous

Director, Office of Group Publishing

Scott Delman

ACM COUNCIL

President

Wendy Hall

Vice-President

Alain Chesnais

Secretary/Treasurer

Barbara Ryder

Past President

Stuart I. Feldman

Chair, SGB Board

Alexander Wolf

Co-Chairs, Publications Board

Ronald Boisvert and Jack Davidson

Members-at-Large

Carlo Ghezzi;

Anthony Joseph;

Mathai Joseph;

Kelly Lyons;

Bruce Maggs;

Mary Lou Soffa;

Fei-Yue Wang

SGB Council Representatives

Joseph A. Konstan;

Robert A. Walker;

Jack Davidson

PUBLICATIONS BOARD

Co-Chairs

Ronald F. Boisvert; Jack Davidson;

Board Members

Nikil Dutt; Carol Hutchins;

Joseph A. Konstan; Ee-Peng Lim;

Catherine McGeoch; M. Tamer Ozsu;

Holly Rushmeier; Vincent Shen;

Mary Lou Soffa; Ricardo Baeza-Yates

ACM U.S. Public Policy Office

Cameron Wilson, Director

1828 L Street, N.W., Suite 800

Washington, DC 20036 USA

T (202) 659-9711; F (202) 667-1066

Computer Science Teachers Association

Chris Stephenson

Executive Director

2 Penn Plaza, Suite 701

New York, NY 10121-0701 USA

T (800) 401-1799; F (541) 687-1840

Association for Computing Machinery (ACM)

2 Penn Plaza, Suite 701

New York, NY 10121-0701 USA

T (212) 869-7440; F (212) 869-0481

COMMUNICATIONS OF THE ACM

Trusted insights for computing's leading professionals.

Communications of the ACM is the leading monthly print and online magazine for the computing and information technology fields. *Communications* is recognized as the most trusted and knowledgeable source of industry information for today's computing professional. *Communications* brings its readership in-depth coverage of emerging areas of computer science, new trends in information technology, and practical applications. Industry leaders use *Communications* as a platform to present and debate various technology implications, public policies, engineering challenges, and market trends. The prestige and unmatched reputation that *Communications of the ACM* enjoys today is built upon a 50-year commitment to high-quality editorial content and a steadfast dedication to advancing the arts, sciences, and applications of information technology.

STAFF

DIRECTOR OF GROUP PUBLISHING

Scott E. Delman

publisher@cacm.acm.org

Executive Editor

Diane Crawford

Managing Editor

Thomas E. Lambert

Senior Editor

Andrew Rosenbloom

Senior Editor/News

Jack Rosenberger

Web Editor

David Roman

Editorial Assistant

Zarina Strakhan

Rights and Permissions

Deborah Cotton

Art Director

Andrij Borys

Associate Art Director

Alicia Kubista

Assistant Art Director

Mia Angelica Balaquiot

Production Manager

Lynn D'Addesio

Director of Media Sales

Jennifer Ruzicka

Marketing & Communications Manager

Brian Hebert

Public Relations Coordinator

Virginia Gold

Publications Assistant

Emily Eng

Columnists

Alok Aggarwal; Phillip G. Armour;

Martin Campbell-Kelly;

Michael Cusumano; Peter J. Denning;

Shane Greenstein; Mark Guzdial;

Peter Harsha; Leah Hoffmann;

Mari Sako; Pamela Samuelson;

Gene Spafford; Cameron Wilson

CONTACT POINTS

Copyright permission

permissions@cacm.acm.org

Calendar items

calendar@cacm.acm.org

Change of address

acmcoa@cacm.acm.org

Letters to the Editor

letters@cacm.acm.org

WEB SITE

<http://cacm.acm.org>

AUTHOR GUIDELINES

<http://cacm.acm.org/guidelines>

ADVERTISING

ACM ADVERTISING DEPARTMENT

2 Penn Plaza, Suite 701, New York, NY

10121-0701

T (212) 869-7440

F (212) 869-0481

Director of Media Sales

Jennifer Ruzicka

jen.ruzicka@hq.acm.org

Media Kit acmm mediasales@acm.org

EDITORIAL BOARD

EDITOR-IN-CHIEF

Moshe Y. Vardi

eic@cacm.acm.org

NEWS

Co-chairs

Marc Najork and Prabhakar Raghavan

Board Members

Brian Bershad; Hsiao-Wuen Hon;

Mei Kobayashi; Rajeev Rastogi;

Jeannette Wing

VIEWPOINTS

Co-chairs

Susanne E. Hambrusch; John Leslie King;

J Strother Moore

Board Members

P. Anandan; William Aspray;

Stefan Bechtold; Judith Bishop;

Stuart I. Feldman; Peter Freeman;

Seymour Goodman; Shane Greenstein;

Mark Guzdial; Richard Heeks;

Rachelle Hollander; Richard Ladner;

Susan Landaur; Carlos Jose Pereira de Lucena;

Beng Chin Ooi; Loren Terveen



PRACTICE

Chair

Stephen Bourne

Board Members

Eric Allman; Charles Beeler; David J. Brown;

Bryan Cantrill; Terry Coatta; Mark Compton;

Stuart Feldman; Benjamin Fried;

Pat Hanrahan; Marshall Kirk McKusick;

George Neville-Neil; Theo Schlossnagle;

Jim Waldo

The Practice section of the CACM

Editorial Board also serves as

the Editorial Board of *queue*.

CONTRIBUTED ARTICLES

Co-chairs

Al Aho and Georg Gottlob

Board Members

Yannis Bakos; Gilles Brassard; Alan Bundy;

Peter Buneman; Ghezzi Carlo;

Andrew Chien; Anja Feldmann;

Blake Ives; James Larus; Igor Markov;

Gail C. Murphy; Shree Nayar; Lionel M. Ni;

Sriram Rajamani; Jennifer Rexford;

Marie-Christine Rousset; Avi Rubin;

Abigail Sellen; Ron Shamir; Marc Snir;

Larry Snyder; Veda Storey;

Manuela Veloso; Michael Vitale;

Wolfgang Wahlster; Andy Chi-Chih Yao;

Willy Zwaenepoel

RESEARCH HIGHLIGHTS

Co-chairs

David A. Patterson and Stuart J. Russell

Board Members

Martin Abadi; Stuart K. Card; Deborah Estrin;

Shafi Goldwasser; Monika Henzinger;

Maurice Herlihy; Dan Huttenlocher;

Norm Jouppi; Andrew B. Kahng;

Gregory Morrisett; Michael Reiter;

Mendel Rosenblum; Ronitt Rubinfeld;

David Salesin; Lawrence K. Saul;

Guy Steele, Jr.; Madhu Sudan;

Gerhard Weikum; Alexander L. Wolf;

Margaret H. Wright

WEB

Co-chairs

James Landay and Greg Linden

Board Members

Gene Golovchinsky; Jason I. Hong;

Jeff Johnson; Wendy E. MacKay



ACM Copyright Notice

Copyright © 2010 by Association for Computing Machinery, Inc. (ACM). Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and full citation on the first page. Copyright for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or fee. Request permission to publish from permissions@acm.org or fax (212) 869-0481.

For other copying of articles that carry a code at the bottom of the first or last page or screen display, copying is permitted provided that the per-copy fee indicated in the code is paid through the Copyright Clearance Center; www.copyright.com.

Subscriptions

An annual subscription cost is included in ACM member dues of \$99 (\$40 of which is allocated to a subscription to *Communications*); for students, cost is included in \$42 dues (\$20 of which is allocated to a *Communications* subscription). A nonmember annual subscription is \$100.

ACM Media Advertising Policy

Communications of the ACM and other ACM Media publications accept advertising in both print and electronic formats. All advertising in ACM Media publications is at the discretion of ACM and is intended to provide financial support for the various activities and services for ACM members. Current Advertising Rates can be found by visiting <http://www.acm-media.org> or by contacting ACM Media Sales at (212) 626-0654.

Single Copies

Single copies of *Communications of the ACM* are available for purchase. Please contact acmhlp@acm.org.

COMMUNICATIONS OF THE ACM

(ISSN 0001-0782) is published monthly by ACM Media, 2 Penn Plaza, Suite 701, New York, NY 10121-0701. Periodicals postage paid at New York, NY 10001, and other mailing offices.

POSTMASTER

Please send address changes to *Communications of the ACM* 2 Penn Plaza, Suite 701 New York, NY 10121-0701 USA



Association for
Computing Machinery



Printed in the U.S.A.



DOI:10.1145/1810891.1810892

Moshe Y. Vardi

Science Has Only Two Legs

Science has been growing new legs of late. The traditional “legs” (or “pillars”) of the scientific method were *theory* and *experimentation*. That was then. In 2005, for example, the U.S.

Presidential Information Technology Advisory Committee issued a report, “Computational Science: Ensuring America’s Competitiveness,” stating: “Together with theory and experimentation, computational science now constitutes the ‘third pillar’ of scientific inquiry, enabling researchers to build and test models of complex phenomena.” The report offered examples such as multi-century climate shifts, multidimensional flight stresses on aircraft, and stellar explosions.

This “third leg” of science has become a standard coin (run a Web search on this phrase!). However, this leg has been recently augmented by yet a “fourth paradigm” (or “leg”) that refers to the usage of advanced computing capabilities to manipulate and explore massive datasets. For example, the decoding of the human genome in 2001 was a triumph of large-scale data analysis. Now science allegedly has four legs, and two of them are computational!

I find myself uncomfortable with science sprouting a new leg every few years. In fact, I believe that science still has only two legs—theory and experimentation. The “four legs” viewpoint seems to imply the scientific method has changed in a fundamental way. I contend it is not the scientific method that has changed, but rather how it is being carried out. Does it matter how many legs science has? I believe it does! It is as important as ever to explain science to the lay public, and it becomes more difficult to explain when it grows a new leg every few years.

Let us consider the first leg: theory.

A scientific theory is an explanatory framework for a body of natural phenomena. A theory can be thought of as a model of reality at a certain level of abstraction. For a theory to be useful, it should explain existing observations as well as generate predictions, that is, suggest new observations. In the physical sciences, theories are typically mathematical in nature, for example, the classical theory of electromagnetism in the form of Maxwell’s Equations. What is often ignored is the fact that any application of a mathematical theory requires computation. To make use of Maxwell’s Equations, for example, we need to solve them in some concrete setting, and that requires computation—symbolic or numeric. Thus, computation has always been an integral part of theory in science.

What has changed is the scale of computation. While once carried out by hand, computation has required over time more advanced machinery. “Doing” theory today requires highly sophisticated computational-science techniques carried out on cutting-edge high-performance computers.

The nature of the theories has also changed. Maxwell’s Equations constitute an elegantly simple model of reality. There is no analogue, however, of Maxwell’s Equations in climate science. The theory in climate science is a highly complex computational model. The only way to apply the theory is via computation. While previous scientific theories were typically framed as mathematical models, today’s theories are often framed as computational mod-

els. In system biology, for example, one often encounters computational models such as Petri Nets and Statecharts, which were developed originally in the context of computer science.

Computation has also always been an integral part of experimentation. Experimentation typically implies carrying out measurements, and the analysis of these measurements has always been computational. Again, what has changed is the scale. The Compact Muon Solenoid experiment at CERN’s Large Hadron Collider generates 40 terabytes of raw data per second, a volume one cannot hope to store and process. Handling such volume requires advanced computation; the first level of data filtering, for example, is carried out on fast, custom hardware using FPGAs. Analyzing the still-massive amount of data that survives various levels of filtering requires sophisticated data-analysis techniques.

So science is still carried out as an ongoing interplay between theory and experimentation. The complexity of both, however, has increased to such a degree that they cannot be carried out without computation. There is no need, therefore, to attach new legs to science. It is doing fine with two legs. At the same time, *computational thinking* (a phrase coined by Jeannette Wing) thoroughly pervades both legs. Computation is the universal enabler of science, supporting both theory and experimentation. Today the two legs of science are thoroughly computational!

Moshe Y. Vardi, EDITOR-IN-CHIEF

SIMONS FOUNDATION

Advancing Research In Basic Science And Mathematics

Program for Mathematics & the Physical Sciences

The Simons Foundation Program for Mathematics and the Physical Sciences seeks to extend the frontiers of basic research. The Program's primary focus is on the theoretical sciences radiating from Mathematics: in particular, the fields of Mathematics, Theoretical Computer Science and Theoretical Physics. Funding for innovative research is available through a peer-reviewed proposal process at regular intervals.

Creation of an Institute for Theoretical Computer Science

Universities and Institutes may apply to create a new Theoretical Computer Science Institute.

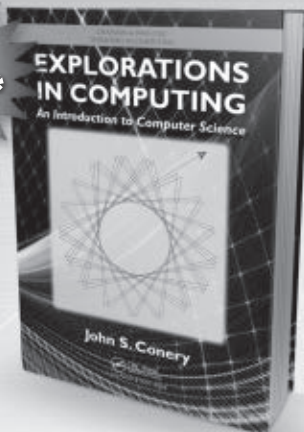
Deadline for Letters of Intent: October 27, 2010

For more information visit

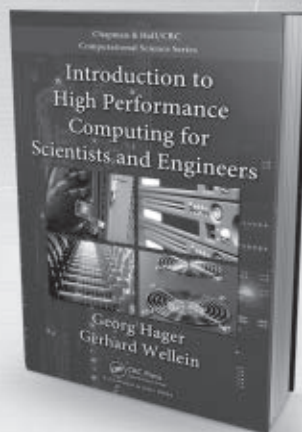
<http://simonsfoundation.org/>

Customize Your CompSci Reading Handbooks for All Levels

Save
20%*

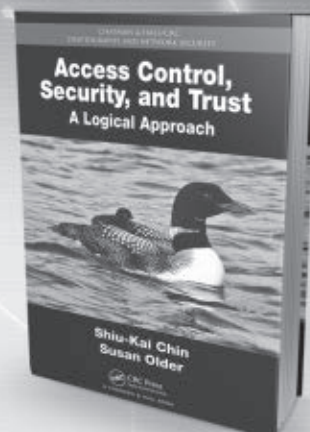


- Includes interactive labs and projects using the open-source language, Ruby
Catalog no. K10640, October 2010
c. 384 pp., ISBN: 978-1-4398-1262-4
~~\$79.95 / £49.99~~ \$63.96 / £39.99



- *"... balanced treatment of the theory, technology, architecture, and software ... I highly recommend this timely book."*
—Jack Dongarra, University of Tennessee

Catalog no. K10600, July 2010
356 pp., Soft Cover, ISBN: 978-1-4398-1192-4
~~\$69.95 / £42.99~~ \$55.96 / £34.39



- Explains how to perform derivations and calculations with mathematical precision

Catalog no. C8628, July 2010
351 pp., ISBN: 978-1-58488-862-8
~~\$89.95 / £57.99~~ \$71.96 / £46.39

Save 20% with promo code 848EM

Receive FREE standard shipping when you order at www.crcpress.com

Offer expires October 30, 2010.

 **CRC Press**
Taylor & Francis Group
A CHAPMAN & HALL BOOK

DOI:10.1145/1810891.1810893

More Than One Way to Annotate Metadata

THE ARTICLE “MANAGING Scientific Data” by Anastasia Ailamaki, Verena Kantere, and Debabrata Dash (June 2010) explained that data generated by research projects is valuable only when annotated with metadata describing the data’s provenance, context, and meaning. However, a given data item can be annotated in more ways than one, for two reasons:

Provenance. A multidisciplinary project can track its progress with basic metadata indicating the provenance of its samples and their associated data. Each data item can also be annotated in a more detailed way through tools particular to the technique used to generate the data item; these annotations are themselves interpretable by people (and software) in the relevant discipline; and

Assumptions. By definition, a research field involves a basic set of concepts used to understand the field but that is not yet agreed upon. Annotations beyond where and when the data was recorded incorporate assumptions that may be contentious among experts.

Data storage and metadata should thus be decoupled. A data repository must be capable of returning any data item stored within it, along with a list of places needed to find the relevant metadata. A metadata repository must be capable of identifying the schema it adheres to and respond to queries about specific data items with relevant annotations.

Decoupling the architecture this way eases development of an ecosystem of repositories and annotation schemas.

Chris Morris, Warrington, U.K.

Authors’ Response:

Separate (and multiple) metadata stores are indeed essential for a number of scientific applications and should be available to user scientists as an option. However, because data queries likely need to combine information held in separate metadata stores, processing them requires appropriate mechanisms for distribution, access control,

and the merging and branching of the stores.

Anastasia Ailamaki, Verena Kantere, Debabrata Dash, Lausanne, Switzerland

No Straw Man in Empirical Research

In his Viewpoint “Is Computer Science Truly Scientific?” (July 2010), Gonzalo Génova would have made a stronger case if he used the words “theoretical” or “conceptual” instead of “speculative” to support his argument against the excessively empirical orientation of much of today’s CS research. The life cycle of scientific ideas generally progresses from the speculative phase in which many candidate ideas are pursued, with only a few surviving to be presented or published as theoretical contributions, often supported by robust analytical models. Journal editors are unlikely to summarily reject contributions making it to this stage because they provide the conjectures and hypotheses that can be tested through rigorous empirically oriented research.

Génova also set up a straw man when he railed against the excesses of verificationism and empiricism. Who would argue against the proposition that credible scientific advances need good empirical research experiments, simulation, proof-of-concept prototype construction, and surveys? Such research needs models and hypotheses that might have begun as speculative conjectures at an earlier point in time.

Naïve empiricism has no place in CS research. Moreover, purely speculative research without adequate analytical foundations is unlikely to help advance CS (or any other) research.

Joseph G. Davis, Sydney, Australia

Author’s Response:

Davis (“credible scientific advances need good empirical research”) and I (“experimentation without the guide of speculative thinking is worthless”) fundamentally agree. When I said “speculative thinking,” I meant “theoretical contributions supported by robust analytical

models,” not freely dancing ideas without purpose.

There may also be slight disagreement regarding empirical validation, the excesses of which I criticized. It is clear that theories about physical phenomena require empirical validation; theories about mathematical objects do not. Many areas in CS deal with conceptual or information objects more akin to mathematical objects than to their physical counterparts. Therefore, requiring empirical validation is out of place here.

Gonzalo Génova, Madrid, Spain

What CS Academics Think They Teach

Poul-Henning Kamp’s article “You’re Doing It Wrong” (July 2010) would have been considerably more valuable and effective if it had been written more professionally and, more important, avoided gross exaggerations. For example, Kamp said the computer architecture depicted in his Figure 7 “is totally bogus today.” Wrong. Though simplistic, it is entirely appropriate as a first architecture for beginning students, most of whom are unable to provide precise definitions even for words like “input” and “output.” Similarly, Kamp saying “It is the only conceptual model used in computer education” cannot be correct.

Alex Simonelis, Montréal

Author’s Response:

Reacting to the article, CS academics have taken offense, protesting the claimed educational deficiencies, while practitioners have confirmed them. I have seen only two reactions saying “We already learned that.” Students evidently do not learn what CS academics think they teach. But the proof is in the pudding; if graduates say “That’s news to me” when reading the article, then the CS academics are doing it wrong.

Poul-Henning Kamp, Slagelse, Denmark

Communications welcomes your opinion. To submit a Letter to the Editor, please limit your comments to 500 words or less and send to letters@cacm.acm.org.

© 2010 ACM 0001-0782/10/0900 \$10.00

The *Communications* Web site, <http://cacm.acm.org>, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.



Follow us on Twitter at <http://twitter.com/blogCACM>

DOI:10.1145/1810891.1810895

<http://cacm.acm.org/blogs/blog-cacm>

Expanding CS Education; Improving Software Development

*Ed H. Chi writes about the social Web's impact on CS education.
Ruben Ortega discusses software and test-driven development.*



Ed H. Chi
**"Time to Rethink
Computer Science
Education:
The (Social) Web
Changes Everything!"**

<http://cacm.acm.org/blogs/blog-cacm/82365>

First, on Monday last week, I read in the news that the U.K. government announced the creation of a new Institute for Web Science. Prime Minister Gordon Brown said 30 million pounds will be used to create this institute to help make " 'public data' public," and act as a bridge between research and business.

Then, this Monday, I read Tim O'Reilly's excellent article on "the State of the Internet Operating System," in which he talked about how the way we organize computing systems in the world is completely different from how we teach computing architectures. He is right. When you think about how we enable a user to type some keywords and get back, say, pictures of a moose, there are a lot of moving parts that all must work together seamlessly. These

components include server farms, IP and caching networks, parallel large-scale data analysis, image and facial recognition algorithms, and maybe location-aware data services. He said the "Internet Operating System" components include search engines, multimedia access, systems relating to user identity and your social network, payment systems, advertising, activity streams, and location. How many universities can say they have experts in all of these areas? These topics are often only covered in computer science departments as either advanced topic courses or, worse, not offered at all.

What do these two pieces of news tell us about the state of the world? There is wide recognition that the Web has changed the world.

"Well, duh!" you say. But there is more....

I read that Rensselaer Polytechnic Institute (RPI) has created the nation's first undergraduate degree in Web Science. The news release said that the students in this interdisciplinary degree program will investigate issues on the Web relating to "security, trust,

privacy, content value." RPI President Shirley Ann Jackson was quoted as saying, "With these new degree programs, students and researchers here at Rensselaer will help to usher in a new era of understanding and study of the Web from its social and economic impacts to the evolution of data." Amen!

When I got my degrees, the university taught compilers, complexity theory, AI, algorithms, operating systems, and databases. While these courses enable me to learn new techniques such as MapReduce, large-scale analytics, visualization, etc., I often feel my education only *equipped* me to prepare for the Web world, but not actually *prepare* me for the Web world. How I wish my undergraduate curriculum included required studies on security and privacy; large-scale data analytics; advanced data-mining techniques; detailed study of recommendation algorithms and systems; as well as HCI research methods like remote user studies, eye tracking, and survey methods.

Am I saying that compilers and theory don't matter anymore? Of course not. They are still excellent academic research pursuits in their own domains, but there might be other new topics that should make it into the curriculum now to better prepare students for a new world. The construction of the new social Web, which is ever changing, requires a different set of skills! The world has changed, and so should the computing science curriculum.

Reader's comment

Sorry, but I couldn't disagree more. This is

not about computer science, it's about how we engineer modern systems.

Computer science is about the fundamentals of creating systems to process information, the basic abstract notions, logic, algebra, algorithms, computability, and computer organization.

It's clear we need to introduce new topics like virtualization, cloud computing, semantic Web, and social networks, but that doesn't change a bit the fundamentals of how computers work.

I went to university in the late 1980s and I've been able to cope with all the changes in those 20 years thanks to a solid formation in the fundamentals.

People need a future-proof education, not a perishable one.

—Pablo Chacin

Blogger's reply

You're making the exact same arguments that some mathematicians and electrical engineers used to make about computer science. How the fundamentals of their areas more than covered the new research directions. Heck, physics folks used to argue, and some still do, that everything derives from their field, so every other field is redundant.

As computer scientists, we have a choice. We can either teach our students compiler design and floating point implementations, or we can teach them machine learning and large-scale data analytics. We can either fold in this new research direction and expand our field, or we can say that it is an application area and let it organically grow until it is so big that it splits into a new department and leaves CS behind. (In fact, I often wonder if it is already too late for us to embrace and extend!)

—Ed H. Chi



Ruben Ortega "How Much Software Testing is Enough?"

<http://cacm.acm.org/blogs/blog-cacm/81546>

Investing in a large amount of software testing can be difficult to justify, particularly for a start-up company. For every line of production software written, an organization should, at minimum, invest an equivalent amount of developer time and number of lines of code to test the created software. The extra effort means that features can take longer to devel-

op and deliver to customers. With the constant pressure of "Deliver Now!" it is very easy to skimp on the amount of testing in an effort to launch sooner. The real difficulty is most developers are good enough that they only do minimal testing to make sure their software works as expected, deploy their software, and move on.

Companies can actually develop software like this for a long time. However, as soon as the software gets beyond a basic complexity level, the number of bugs that creep back in via regression or untested use cases will result in an unstable application. At this point the company is compelled to either (a) stop development, and add the regression tests they failed to do earlier, or (b) continue a bad pattern where a team of software testers chase regression bugs and add test suites for the previous version of software, while other developers create the next set of features (and bugs) concurrently. Both these patterns are flawed because the time they take to fix the issue is longer than it would have taken had the tests been created continuously.

Test-driven development, an Extreme Programming practice, is arguably one of the best ways to help ensure that the created software always has a truss to test it. The basic methodology is to create the test suite first, have it fail, and then create the methods that will get it to pass successfully. This helps to ensure that there is at least one test case for each method created by the developer who wrote the software. By having the testing harness developed concurrently with the software you will have placed the responsibility of testing on the developer who created the feature. This means the company saves time in overall development because the tests are created by people knowledgeable about what needs to be tested, and software can be tested continuously on every source code commit allowing for deployment on demand.

This leaves one critical hole in the testing process. How good is the test suite that the developer created? This is the point where I put on a pragmatic's hat. If your organization already has the discipline to test every method of your software, you should

probably ask the developer to just test the "basic" behavior and allow for extending the test suite if a new bug emerges. The purpose of the test harness is to make sure the software works given the known assumptions of the software, and having them re-tested on every check-in and deployment helps build confidence that you are deploying correct software. The most dramatic bugs I have seen, with or without a test harness, have generally happened when an unanticipated event occurred, and testing against the unknowable is difficult.

My favorite story about unanticipated bugs that would have been helped by having a test harness in place occurred early in my tenure at Amazon. It was a bug I affectionately call "Karmic Revenge." The site was crashing on a subset of Amazon's book catalog, and it happened disturbingly frequently on the search results page. I was called in to identify the bug. (For those coders in the audience: I discovered that a data structure we were using was referencing an array at location offset of [-1] which was causing the software to crash.) The catalog software had changed recently such that the number -1 was a flag that no data was available. Unfortunately, this knowledge hadn't propagated through the search software. The "Karmic Revenge" was the book that displayed the problem was about "Memory Management in C." Additionally, for the superstitious, the date the bug was identified, debugged, and fixed was Friday, February 13, 1998. Some bugs you just can't forget.

Had a test harness been in place, perhaps this bug would have never made it to the production site. Or if the bug had made it to the site, then once found, a new test would have been added to the test harness to prevent future occurrences. However, the structure didn't exist either in the code or at the organizational level. Better patterns of development will always reduce the likelihood of this error occurring and reoccurring. ■

Ed H. Chi is a research manager at Palo Alto Research Center. Ruben Ortega is an engineering director at Google.

© 2010 ACM 0001-0782/10/0900 \$10.00

ACM, Advancing Computing as a Science and a Profession



Dear Colleague,

The power of computing technology continues to drive innovation to all corners of the globe, bringing with it opportunities for economic development and job growth. ACM is ideally positioned to help computing professionals worldwide stay competitive in this dynamic community.

ACM provides invaluable member benefits to help you advance your career and achieve success in your chosen specialty. Our international presence continues to expand and we have extended our online resources to serve needs that span all generations of computing practitioners, educators, researchers, and students.

ACM conferences, publications, educational efforts, recognition programs, digital resources, and diversity initiatives are defining the computing profession and empowering the computing professional.

This year we are launching Tech Packs, integrated learning packages on current technical topics created and reviewed by expert ACM members. The Tech Pack core is an annotated bibliography of resources from the renowned ACM Digital Library – articles from journals, magazines, conference proceedings, Special Interest Group newsletters, videos, etc. – and selections from our many online books and courses, as well as non-ACM resources where appropriate.

BY BECOMING AN ACM MEMBER YOU RECEIVE:

Timely access to relevant information

Communications of the ACM magazine • ACM Tech Packs • TechNews email digest • Technical Interest Alerts and ACM Bulletins • ACM Journals and magazines at member rates • Full access to the unique website for practitioners • ACM SIG conference discounts • the optional ACM Digital Library

Resources that will enhance your career and follow you to new positions

Career & Job Center • online books from Safari® featuring O'Reilly and Books24x7® • online courses in multiple languages • virtual labs • e-mentoring services • CareerNews email digest • access to ACM's 34 Special Interest Groups • an acm.org email forwarding address with spam filtering

ACM's worldwide network of more than 97,000 members ranges from students to seasoned professionals and includes many renowned leaders in the field. ACM members get access to this network and the advantages that come from their expertise to keep you at the forefront of the technology world.

Please take a moment to consider the value of an ACM membership for your career and your future in the dynamic computing profession.

Sincerely,

Alain Chouais
President
Association for Computing Machinery



Association for
Computing Machinery

Advancing Computing as a Science & Profession



Association for
Computing Machinery

Advancing Computing as a Science & Profession

membership application & digital library order form

Priority Code: AD10

You can join ACM in several easy ways:

Online
<http://www.acm.org/join>

Phone
+1-800-342-6626 (US & Canada)
+1-212-626-0500 (Global)

Fax
+1-212-944-1318

Or, complete this application and return with payment via postal mail

Special rates for residents of developing countries:
<http://www.acm.org/membership/L2-3/>

Special rates for members of sister societies:
<http://www.acm.org/membership/dues.html>

Please print clearly

Name _____

Address _____

City _____ State/Province _____ Postal code/Zip _____

Country _____ E-mail address _____

Area code & Daytime phone _____ Fax _____ Member number, if applicable _____

Purposes of ACM

ACM is dedicated to:

- 1) advancing the art, science, engineering, and application of information technology
- 2) fostering the open interchange of information to serve both professionals and the public
- 3) promoting the highest professional and ethics standards

I agree with the Purposes of ACM:

Signature _____

ACM Code of Ethics:
<http://www.acm.org/serving/ethics.html>

choose one membership option:

PROFESSIONAL MEMBERSHIP:

- ☐ ACM Professional Membership: \$99 USD
- ☐ ACM Professional Membership plus the ACM Digital Library: \$198 USD (\$99 dues + \$99 DL)
- ☐ ACM Digital Library: \$99 USD (must be an ACM member)

STUDENT MEMBERSHIP:

- ☐ ACM Student Membership: \$19 USD
- ☐ ACM Student Membership plus the ACM Digital Library: \$42 USD
- ☐ ACM Student Membership PLUS Print CACM Magazine: \$42 USD
- ☐ ACM Student Membership w/Digital Library PLUS Print CACM Magazine: \$62 USD

All new ACM members will receive an
ACM membership card.

For more information, please visit us at www.acm.org

Professional membership dues include \$40 toward a subscription to *Communications of the ACM*. Member dues, subscriptions, and optional contributions are tax-deductible under certain circumstances. Please consult with your tax advisor.

RETURN COMPLETED APPLICATION TO:

Association for Computing Machinery, Inc.
General Post Office
P.O. Box 30777
New York, NY 10087-0777

Questions? E-mail us at acmhelp@acm.org
Or call +1-800-342-6626 to speak to a live representative

Satisfaction Guaranteed!

payment:

Payment must accompany application. If paying by check or money order, make payable to ACM, Inc. in US dollars or foreign currency at current exchange rate.

☐ Visa/MasterCard ☐ American Express ☐ Check/money order

☐ Professional Member Dues (\$99 or \$198) \$ _____

☐ ACM Digital Library (\$99) \$ _____

☐ Student Member Dues (\$19, \$42, or \$62) \$ _____

Total Amount Due \$ _____

Card # _____ Expiration date _____

Signature _____



DOI:10.1145/1810891.1810896

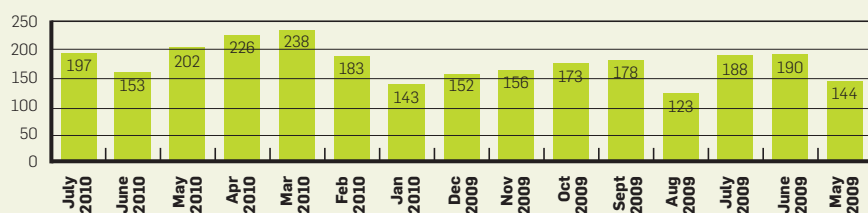
David Roman

More *Communications*

Numbers give meaning to the promises on *Communications* Web site to deliver timely, substantive content that complements the magazine's peer-reviewed material and makes *Communications* a valued component of ACM membership. The following data illustrates a portion of the content available only at cacm.acm.org.^a

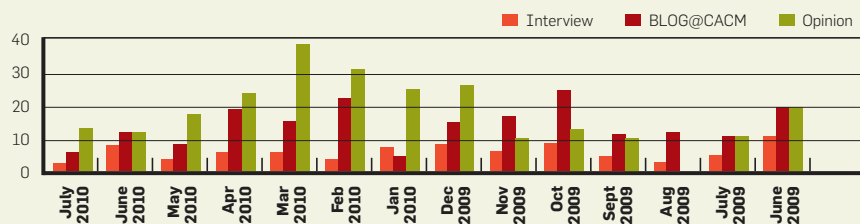
News Coverage. News from industry, academia, ACM TechNews, the technical and mainstream media, plus original articles helps drive the site with fresh content every business day.

Number of monthly news articles posted exclusively on cacm.acm.org/.



Points of View. Expert contributors make the BLOG@CACM (<http://cacm.acm.org/blogs/blog-cacm>) one of the most popular features of the site. Opinion articles (<http://cacm.acm.org/opinion/articles>) and Interviews (<http://cacm.acm.org/opinion/interviews>) also inspire lively discussions.

Number of opinion pieces posted each month.



Professional Help. The Careers page (<http://cacm.acm.org/careers>) provides information and advice on employment and professional issues.

Number of career-related articles.



^a Some numbers for July 2010 are incomplete due to production deadlines.

ACM Member News

ANDREAS MOSHOVOS WINS ACM SIGARCH'S MAURICE WILKES AWARD



The ACM Special Interest Group on Computer Architecture (SIGARCH) presented the

2010 Maurice Wilkes Award to Andreas Moshovos, an associate professor in the computer engineering group at the University of Toronto, for his contributions to the development of memory dependence prediction. This technique, used by high-performance microprocessors that execute memory access operations, provides many applications in boosting memory system performance and reducing processor design complexity. "Surprised, humbled, and honored" is how Moshovos describes his reaction to winning the Maurice Wilkes award. "I was fortunate to work at the University of Wisconsin-Madison with professor Guri Sohi, who at the time was working on Multiscalar, a forward-looking architecture," Moshovos said in an email interview. "Multiscalar exposed problems that were not obvious at the time. That enabled Scott Breach and T.N. Vijaykumar, my two collaborators on the early memory dependence prediction work, and I to see these problems and try to find solutions."

Moshovos' current research involves design challenges for the ever-growing gap between processor and memory performance. "Parallel programming is becoming a necessity. Much of our work is on mechanisms to allow future multi-core processors to efficiently support parallel programs. Our snoop filtering techniques reduce energy while making it simpler to build small- to medium-scale multi-cores. This work has already influenced commercial designs. With colleagues from IBM T. J. Watson, we developed tagless directories—an area- and power-efficient solution—that targets larger-scale systems."

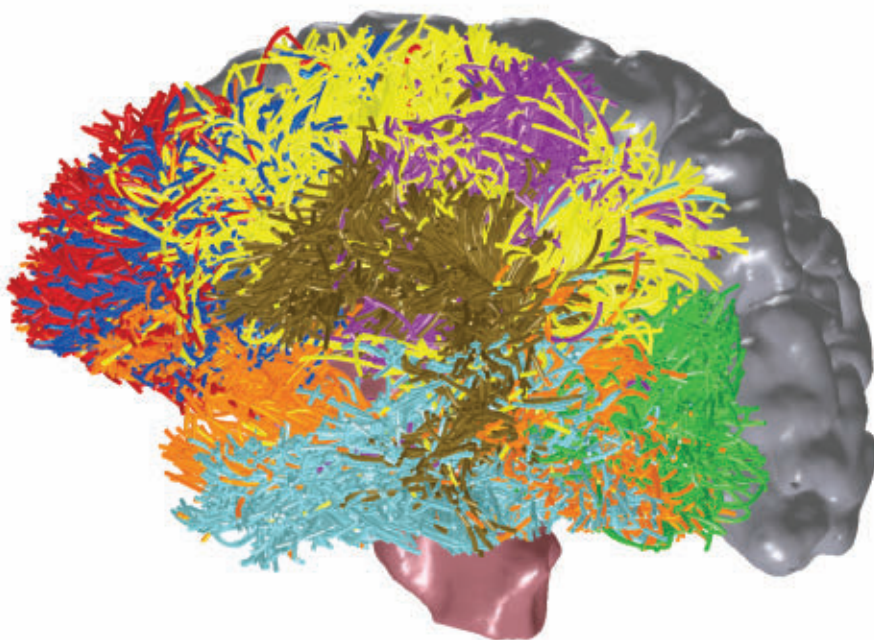
—Jack Rosenberger

Brains and Bytes

Computational neuroscientists are learning that the brain is like a computer, except when it isn't.

THE IDEA THAT the human brain can be thought of as a fancy computer has existed as long as there have been computers. In a simple way, the analogy makes sense: A brain takes in information and manipulates it to produce desirable output in the form of physical actions or, more abstractly, plans and ideas. Within the brain information moves in the form of electrical signals zipping through and among neurons, the nerve cells that form the elementary signal processing units of biological systems. But the likeness between brains and electronic computers only goes so far. Neurons, as well as the connections between them, work in unpredictable and probabilistic ways; they are not simple gates, switches, and wires. For researchers working in computational neuroscience, the looming puzzle is to understand how a brain built from fundamentally unreliable components can so reliably perform tasks that digital computers have barely begun to crack.

The brain of a human adult contains approximately 100 billion neurons, each of which has an average of several thousand connections to other neurons. As neuroscientists have long realized, it's the complex connectivity as much as the sheer number of



In collaboration with researchers from Stanford University, IBM scientists have developed an algorithm that uses the Blue Gene supercomputing architecture to noninvasively measure and map the connections between all cortical and sub-cortical locations within the human brain.

neurons that make understanding brain function such a daunting task. To that end, researchers study individual neurons and small collections of them with the aim of building an explanation for neural activity from the ground up. But that task is also far from simple.

Typically, a neuron constantly receives signals from thousands of oth-

er neurons. Whether a neuron fires and thereby passes along a signal to the thousands of connecting neurons depends on the nature of the input it receives. Neural firing and signal transmission are chemical processes and the fundamental reason they are not completely reliable, says Hsi-Ping Wang, a researcher at the Salk Institute for Biological Studies, is that “biology

is messy. Molecules have to bind and unbind, and chemical and signal elements have to mix and diffuse.”

Nature bypasses this messiness in part by resorting to statistical methods. For example, Wang and his colleagues combined recordings of in vivo neural activity, with a computer simulation of neuron function in the visual cortex of a cat, to show that neurons fired most reliably when they were stimulated by the almost simultaneous arrival of approximately 30 input signals. With fewer than 20 signals arriving at once, the neuron was significantly less likely to fire, but the simultaneous arrival of more than 40 signals brought no gain in the reliability of the output signal. It’s not inconceivable that nature could make more reliable neurons, says Wang, but “there is a cost to making things perfectly reliable—the brain can’t afford to expend resources for that additional perfection when slightly probabilistic results are good enough.”

Neuron firing is not the brain’s only probabilistic element. Signals pass from one neuron to another through biomolecular junctions called synapses. In a phenomenon known as plasticity, the likelihood that a synapse will transmit a spike to the next neuron can vary depending on the rate and timing of the signals it receives. Therefore, synapses are not mere passive transmitters but are information processors in their own right, sending on a transformed version of the received signals.

These complications cast light on the usefulness of thinking the brain is

The current puzzle is to understand how a brain built from fundamentally unreliable components can reliably perform tasks that digital computers have barely begun to crack.

similar to a massively parallel supercomputer, with many discrete processors exchanging information back and forth. Information processing by the brain crucially depends on loops and feedbacks that operate down to the level of individual neurons, says Larry Abbott, co-director of the Center for Theoretical Neuroscience at Columbia University, and that aspect makes the nature of the processing exceedingly difficult to analyze. “The brain is stochastic and dynamical,” Abbott says, “but we can control it to an incredible degree. The mystery is how.”

As an illustrative example, Jim Bednar, a lecturer at the Institute for Adap-

tive and Neural Computation School of Informatics at the University of Edinburgh, considers what happens when a person catches a flying ball. It’s tempting—but wrong—to suppose the visual area of the brain figures out the ball’s path, then issues precise instructions to the muscles of the arm and hand so that they move to just the right place to catch the ball. In reality, neural processes are messy and stochastic all the way from the visual cortex down to our fingertips, with multiple and complex signals going back and forth as we try to make the hand and ball intersect. In this system, Bednar says, “the smartness is everywhere at once.”

It’s also important to realize the operation of the brain cannot be overly sensitive to the behavior of individual neurons, Bednar adds, if for no other reason than that “neurons die all the time.” What matters, he says, must be the collective behavior of thousands of neurons or more, and on those scales the vagaries of individual neural operation may be insignificant. He regards the brain as having a type of computational workspace that simultaneously juggles many possible solutions to a given problem, and relies on numerous feedbacks to strengthen the appropriate solution and diminish unwanted ones—although how that process works, he admits, no one yet knows.

Empirical Knowledge

Alexandre Pouget, associate professor of brain and cognitive sciences and bioengineering at Rochester University,

Obituary

Carl Adam Petri, 1926–2010

Carl Adam Petri, a German mathematical and computer scientist who invented Petri nets, a modeling language used to describe and document concurrent processes through the use of graph-based structures, died on July 2 at age 83.

Petri is considered a pioneer in advancing the fields of parallel computing and distributed computing. He also played a key role in developing methods of analysis for complex systems and workflow management.

“In the 1960s he had already developed ideas about modeling, distributed systems, and computers as a communication medium,” notes Wolfgang Reisig, a professor at the Computer Science Institute, Humboldt-University of Berlin. “The work was remarkable because it took decades for the computing science mainstream to accept his early vision.”

Born in Leipzig, Germany, Petri documented Petri nets as part of his dissertation, “Communication

with Automata,” in 1962. He taught at several universities in Germany, and served as scientific director for a research institute at the National Center for Mathematics and Computing, a research lab near Bonn.

In 1966, Petri was the recipient of the Werner von Siemens Ring, a prestigious German award in technical sciences. In 2003, he received the Order of the Netherlands Lion award and was honored by the Queen of the Netherlands.

In 2007, Petri received the Academy Gold Medal of Honor, a lifetime achievement award presented by the Academy of Transdisciplinary Learning and Advanced Studies. IEEE presented Petri with its Computer Pioneer Award in 2008 for establishing Petri net theory, “which not only was cited by hundreds of thousands of scientific publications but also significantly advanced the fields of parallel and distributed computing.”

—Samuel Greengard

uses empirical knowledge of neuron function to build theoretical and computational models of the brain. Noting that neural circuitry exhibits common features through much of the cortex, he argues that the brain relies on one or a handful of general computational principles to process information. In particular, Pouget believes the brain represents information in the form of probability distributions and employs methods of statistical sampling and inference to generate solutions from a wide range of constantly changing sensory data. If this is true, the brain does not perform exact, deterministic calculations as a digital computer does, but reliably gets “good enough answers in a short time,” says Pouget.

To explore these computational issues, Pouget adopts a cautious attitude to the question of how important it is to know in precise detail what individual neurons do. For example, the exact timing of output spikes may vary from one neuron to another in a given situation, but the spike rate may be far more consistent—and may be the property that a computational method depends on. In their modeling, Pouget says he and his colleagues “simplify neurons as much as possible. We add features when we understand their computational role; we don’t add details just for the sake of adding them.”

The distinction between understanding individual neurons and understanding the computational capacity of large systems of neurons is responsible for “a huge schism in the modeling community,” Bednar says. That schism came into the open last year when Dharmendra Modha, manager of cognitive computing at IBM Almaden Research Center, and colleagues reported a simulation on an IBM Blue Gene supercomputer of a cortex with a billion neurons and 10 trillion synaptic connections—a scale that corresponds, the authors say, to the size of a cat’s brain. This claim drew a public rebuke from Henry Markram, director of the Blue Brain Project at the École Polytechnique Fédérale de Lausanne, which is using supercomputers to simulate neurons in true biological detail. Markram charged that the neurons in Modha’s simulation were so oversimplified as to have little value in helping to un-

derstand a real neural system. In contrast, the Blue Brain project exhausts the capacity of a Blue Gene supercomputer in modeling just some tens of thousands of neurons.

“Both those approaches are important,” says Abbott, although he thinks the task of understanding brain computation is better tackled by starting with large systems of simplified neurons. But the big-picture strategy raises another question: In simplifying neurons to construct simulations of large brains, how much can you leave out and still obtain meaningful results? Large-scale simulations such as the one by Modha and his colleagues don’t, thus far, do anything close to modeling specific brain functions. If such projects “could model realistic sleep, that would be a huge achievement,” says Pouget.

With today’s currently available computing power, computational neuroscientists must choose between modeling large systems rather crudely or small systems more realistically. It’s not yet clear, Abbott says, where on that spectrum lies the sweet spot that would best reveal how the brain performs its basic functions. **C**

Further Reading

Abbott, L.F.

Theoretical neuroscience rising, *Neuron* 60, 3, Nov. 6, 2008.

Ananthanarayanan, R., Esser, S.K., Simon, H.D., and Modha, D.S.

The cat is out of the bag: cortical simulations with 109 neurons, 1013 synapses, Proc. Conf. on High Performance Computing, Networking, Storage, and Analysis, Portland, OR, Nov. 14–19, 2009.

Miikkulainen, R., Bednar, J.A., Choe, Y., and Sirosh, J.

Computational Maps in the Visual Cortex. Springer, New York, NY, 2005.

Pouget, A., Deneuve, S., and Duhamel, J.-R.

A computational perspective on the neural basis of multisensory spatial representations, *Nature Reviews Neuroscience* 3, Sept. 2002.

Wang, H.-P., Spencer, D., Fellous, J.-M., and Sejnowski, T.J.

Synchrony of thalamocortical inputs maximizes cortical reliability, *Science* 328, 106, April 2, 2010.

David Lindley is a science writer and author based in Alexandria, VA.

© 2010 ACM 0001-0782/10/0900 \$10.00

Security

New Passwords Approach

To prevent dictionary attacks by hackers, passwords have become increasingly complex. And while administrators concoct ever-stricter criteria for passwords that add millions of more passwords and make dictionary attacks increasingly difficult, users want passwords to be more memorable than a long string of random, case-sensitive characters. Now, a pair of Microsoft researchers and a Harvard professor have a simple solution that’s easier to remember, and much less attractive to criminal hackers.

In a paper titled “Popularity is Everything: A New Approach to Protecting Passwords From Statistical-Guessing Attacks,” which was presented at the Hot Topics in Security conference in Washington, D.C. last month, Microsoft researchers Stuart Schechter and Cormac Herley and Harvard computer science professor Michael Mitzenmacher have proposed a system that allows users to choose any password they want, so long as it’s not popular among other users. “Each time a user chooses a new password, the occurrence of that password is counted by a probabilistic data structure known as a count-min sketch,” said Schechter and Herley in a joint email message. “We chose to use the count-min sketch because it allows us to identify dangerous[ly] popular passwords, yet do so while limiting the amount of information that could be leveraged by attackers if they captured it.”

After a small number of users choose the same password, no new users may use that word. This deprives attackers of common passwords that allow them to break into a significant number of online accounts, designers can keep rules simple for password selection, and users don’t need to study those rules.

This new approach is similar to one adopted by Twitter, which, following an online password-guessing attack, now forbids 390 of the most common passwords, including “ABCD,” “1234,” and “password.”

—Phil Scott

Cycling Through Data

Sensor-equipped bicycles are providing valuable data to cyclists, city planners, and computer scientists.

TRADITIONALLY, THE INTERNET has been viewed as a collection of more or less stationary machines wired to a network. But with the popularity of smartphones and their embedded sensors—a camera, a microphone, an accelerometer to determine the phone's orientation, and a global positioning system (GPS) locator—the power of computing is spreading beyond the desktop. Last year an executive from telecom giant Ericsson predicted that 50 billion devices will be connected by 2020, leading to an “Internet of things” in which everyday objects become part of the network, gathering new types of data and creating possibilities that didn't previously exist.

One activity where the Internet may soon tie into non-cyberspace life is the world of cycling. The Copenhagen Wheel, a project of the SENSEable City Lab, part of the Department of Urban Studies and Planning at the Massachusetts Institute of Technology (MIT), places environmental sensors on the rear wheel of a bicycle, providing information that could add value to the rider's experience while also giving valuable data to city planners. Biketastic, a project of the Center for Embedded Networked Sensing at the University of California at Los Angeles, also enables cyclists to share their experiences, using just the smartphone's microphone and accelerometer to measure route conditions. The idea is to help cyclists find the safest, most efficient, and most enjoyable routes through the metropolis.

The Copenhagen Wheel was introduced at the Copenhagen Climate Conference last December, and MIT is in talks with city officials to make it available there as part of Copenhagen's campaign to increase bicycle commuting from about 35% to 50% of its 500,000 citizens. The Wheel, which replaces the rear wheel of a standard coaster-brake bicycle, contains a small motor and 16



With the Copenhagen Wheel, an ordinary bicycle is transformed into a hybrid electronic vehicle that records road conditions, traffic congestion, and pollution levels in real time.

lithium-polymer batteries, recharged by regenerative braking. When the rider needs a boost—because she's climbing a steep hill, for instance—the motor kicks in to assist. When she brakes, the energy is stored in the batteries.

The hub of the wheel also contains electronics to control the motor and measure torque, and a Bluetooth connection to communicate with a smartphone. A cradle on the handlebars holds the smartphone, which runs an app that allows the rider to select one of three modes, providing extra assistance for a rider who wants to get to work without much sweat, or increasing resistance to put the rider through a workout and recharge the batteries. The rider can also switch gears through the phone.

But what makes the Wheel computationally interesting is the addition of sensors and networking, “turning a bike from something that gets you from A to B into a smart network object,” says Christine Outram, an MIT research associate who leads the project. The hub contains sensors that measure carbon

monoxide, nitrogen oxides, temperature, humidity, and noise, taking readings every two seconds.

Not all that data is displayed during the ride—it could be too distracting. Generally, the smartphone will alert the rider only when she reaches a personal best or when a friend is within a designated proximity. The phone can store the data and let the user review it later.

The data can also be transmitted to a city-owned server, and Xiaoji Chen, an MIT graduate student in architecture who works with the SENSEable City Lab, has created a program that will enable city planners to visualize the data. A map displays the incoming data from each rider as a series of color-coded spikes. A viewer can watch, say, the noise level as a rider travels through the city, with older data slowly fading from view to keep the data current.

“If we have enough data, we can see how this changes over weeks or months or years,” Chen says. The data can be cross-referenced with information about land use at various points along

the routes. “We can see how this pollution level is related to weather, or to policies promoting bicycles,” she says.

The data could help city planners identify urban heat islands, where an abundance of asphalt and concrete artificially raise temperatures. It could also pinpoint areas suffering from noise pollution or a concentration of exhaust fumes. By measuring the speed of bicyclists and how often they stop, the system could alert both traffic planners and cyclists to areas of traffic congestion. And measurements from the smartphone’s accelerometer could call attention to potholes or other potentially dangerous street conditions.

Copenhagen currently has three fixed environmental sensors in the city, one at street level and two about three stories high. Because there will be more of them and they will be at street level, sensors on the bikes will provide a lot more point data. It will only take about 100 cyclists equipped with the Wheel to get good coverage of a two-kilometer-square downtown area, says Outram.

Cyclists tapping into the network can also see historical data not only for their own bike routes but for those used by others. That may allow them to choose a route that is quieter or less polluted. It also lets them interact with their fellow cyclists if they choose to, meeting en route for a break or to ride together. “We’re trying to make some of the connections through Facebook or other social networking sites—these virtual connections—physical,” Outram says.

Los Angeles’ Biketastic

While European metropolises like Copenhagen tend to have compact downtowns conducive to bike riding, sprawling Los Angeles is very much geared toward cars. But biking is still popular there, and the Biketastic project aims to make it easier for Los Angelenos to find safe and pleasant routes, both for commuting and recreational rides. University of California at Los Angeles (UCLA) researchers designed an application for Android phones and conducted a two-week pilot project last fall.

Riders launch the Biketastic application when they mount their bikes, and the phone uses GPS data to trace the route and measure the cyclist’s speed. The app asks if the phone is in a bag or out in the open, so it knows whether

About 450 cyclists have registered for the Biketastic Web site and mapped out almost 1,400 routes in Los Angeles.

it can use the phone’s microphone to measure noise. If the cyclist mounts the phone on the bicycle, the phone figures out its orientation and uses the accelerometer to measure road roughness. The rider can take pictures of interesting landmarks or dangers such as potholes. All the data is uploaded to a Web site, which overlays the information on Google Maps, adds information about elevation, and allows cyclists to include tags and descriptions.

Rather than design new devices, Sashank Reddy, the Ph.D. student at UCLA who headed the Biketastic project, wants to take advantage of sensors people already carry with them. “Our purpose is to understand how we can use the sensors, what are the algorithms to clean up the data, and what are appropriate visualization techniques,” he says.

About 450 users have registered for the Biketastic Web site and mapped out almost 1,400 routes. While the app is available to Android phone users, it probably takes a minimum number of people in a city to be useful, Reddy says, and he’s not focused on expanding the project, though he wouldn’t mind seeing someone else commercialize products or services based on his work. Cycling advocacy groups and city planners have asked him about how they might use his techniques, but nothing has progressed beyond the pilot stage.

Ron Milam, a consultant for planners of environmentally friendly projects and a writer about biking in Los Angeles on his BikeSage blog, participated in the pilot project and got a better sense of the speed and distances he was riding. He thinks Biketastic can be particularly useful by providing people with routes that others have discovered,

and thus make it easier for people to shift from driving to cycling for some of their trips. “Having this information around could not only inspire people to ride a bike but also give them some really concrete information on places they may want to go,” Milam says. “The perception is you can’t ride a bike or walk in L.A., but the reality is more people are choosing to do that here.”

Meanwhile, Outram’s team is working on making the Copenhagen Wheel more compact, and hopes to commercialize it within the year. They plan to first sell it to cities, for use on fleets of bicycles used by police or traffic enforcement officers. In addition to Copenhagen, Los Angeles, Mexico City, Sydney, and Wellington, New Zealand have expressed interest. Once the cities get the system operating, individual cyclists would be able to start buying in.

Outram sees the Wheel as just one example of how the growing ubiquity of computing, sensing, and communication can improve both individual lives and society as a whole. “The broader vision of the lab is to ask, ‘What is the future of living with technology going to be?’ ” she says. “In 10 to 15 years, we’ll be able to have this ‘Internet of things.’ We want to explore possibilities about how we live with technology in a kind of human way.” 

Further Reading

Hirshberg

The Copenhagen wheel, http://www.youtube.com/watch?v=Do3lxv_ekUo

Iveson, K.

Too public or too private? The politics of privacy in the real-time city, First International Forum on the Application and Management of Personal Electronic Information, Cambridge, MA, October 12–13, 2009.

Ratti, C., Pulselli, R.M., Williams, S., and Frenchman, D.

Mobile landscapes: using location data from cell phones for urban analysis, *Environment and Planning B* 33, 5, 2006.

Reddy, S., Shilton, K., Denisov, G., Cenizal, C., Estrin, D., and Srivastava, M.

Biketastic: sensing and mapping for better biking, ACM Conference on Human Factors in Computing Systems, Atlanta, GA, April 10–15, 2010.

Neil Savage is science and technology writer based in Lowell, MA.

© 2010 ACM 0001-0782/10/0900 \$10.00

Degrees, Distance, and Dollars

The Internet is making higher education accessible to a whole new class of students—but not necessarily at a lower cost.

PODCASTING, HIGH-SPEED INTERNET, email, message boards—the technology for distance learning has made it less and less necessary for students to go to college the old-fashioned way. Yet, the demand for higher education continues to rise at double-digit rates, boosting the number of students taking one or more online courses in the U.S. in the fall of 2008 to 2.4 million, up from 1.6 million in 2002, according to the most recent survey by the Sloan Consortium, an organization supporting online education.

These numbers do not include the free non-credit courses available through iTunes U, which offers 250,000 free lectures from more than 600 schools, including Yale and Massachusetts Institute of Technology. Yet while elite schools are reaching the masses as a philanthropic gesture, they tend to avoid granting more degrees. “Your Stanfords and Columbias and NYUs and Boston Colleges of the world—they have terrific incentives not to grow,” says Guilbert C. Hentschke, a professor at the University of Southern California’s (USC’s) Rossier School of Education. That’s because exclusive schools are, by definition, highly selective—and admitting more students would dilute their brands.

A similar dynamic works in other traditional universities, as well. A school’s *U.S. News & World Report* annual ranking depends in part on teacher salaries and per-student spending, so going online and reducing costs can tarnish a school’s image. If public universities graduate many students who have taken online courses, it’s only because the schools are by far the largest sector in American higher education, Hentschke says. State funding has kept most of them from going online in a substantial way, though severe budget



The range of CS courses offered in online education covers everything from introductory programming to graduate-level courses in database theory.

cuts in recent months are prompting the University of California, Berkeley and Rutgers University, among others, to consider online instruction to help fill budget gaps.

Even so, the biggest growth in online schooling has been among for-profit universities, which includes obscure institutions and more familiar names like Kaplan, DeVry University, and the biggest gorilla of all, the University of Phoenix, which currently has 455,600 students, more than half of whom take at least some courses online. As public community colleges turn away tens of thousands of students each year, they create a huge opportunity for for-profits offering associate degrees and higher. The online schools lure students with Internet ads touting instant enrollment and 24/7 access. And whereas traditional universities use tuition from large lecture classes to

cover losses from costlier or undersubscribed programs, for-profits can aim precisely where the money is, focusing on degrees in computer science (especially IT), business, health care, education, and other marketable fields.

Most of the students who flock to online programs are nontraditional; their time and locale is constrained by jobs, military service, and dependent children. “Online education is not only more convenient, but for some students it’s the only option they’ve got,” says Hentschke.

Education in computer science is a case in point. The National Science Foundation wants to increase the number of advanced placement CS teachers to 10,000 by 2015, which is five times today’s number. Current undergraduates alone aren’t likely to meet that demand, but by taking online classes after work, other candidates, such as math teachers, can branch out into teaching CS, suggests Mark Guzdial, a professor at George Institute of Technology and expert in computer science education. Similarly, if female managers in tech firms hit the glass ceiling in part by not finding time to learn the latest tools, as an Anita Borg Institute study suggests, then being able to take classes from home should help close the gender gap in upper management. “These two audiences are poorly served by face-to-face CS courses, but may be well served by distance learning,” Guzdial says.

Online education in computer science varies in quality, but the range of offerings is impressive, covering everything from introductory programming and Microsoft certification to graduate-level courses in database theory, network security, and human-computer interaction.

Convenience at a Price

You might expect online courses to

also be cheaper, but that is rarely the case. Although online students save time, living expenses, and transportation costs, they typically pay at least as much in tuition as they would for a traditional education. The University of Phoenix, for example, charges the same for both formats, and according to the College Board the average sticker price at a for-profit university is about \$14,000 for the 2009–2010 academic year.

Why the high price even online? Some education experts contend that good instruction is always labor-intensive. “I could set up an online course and have a thousand students and teach it myself, but what’s the quality going to be?” says Donald Heller, who directs the Center for the Study of Higher Education at Pennsylvania State University. It is true that technology enables a small team to design a course and a lower-paid army of instructors to deliver it, grade papers, and interact with students, but that is not very different from what traditional colleges have been doing for decades, Heller argues. Diane Harley, a University of California, Berkeley anthropologist who directs the university’s Higher Education in the Digital Age research project, agrees. “It’s not cheap to produce high-quality online courses from soup to nuts,” she says, quoting the oft-cited \$1 million per state-of-the-art course such as those produced by Carnegie Mellon University’s Open Learning Initiative.

Nonetheless, because schools can add students without erecting new buildings, a school’s costs for each additional student can be quite small—low enough that a company called Straighterline profitably sells basics on algebra and English composition for just \$99 per month, plus \$39 per course. Although it doesn’t grant degrees, Straighterline grades coursework and issues transcripts that students can turn into credits at the colleges where they are enrolled.

But this low-priced model remains the exception in online education, where for a host of reasons schools have not passed their savings on to students. In fact, sometimes an online degree costs more than its brick-and-mortar equivalent—a price premium not just for convenience, but for has-

tle-free admissions, suggests Vicky Phillips, founder and chief analyst of GetEducated.com, a watchdog group for online learning. “People research schools with online MBA programs and find out that Indiana University has a price that’s one-third of the University of Phoenix’s price, and then they find out they have to take the GRE and GMAT and 12 prerequisite courses [for Indiana University]. Welcome to the age of ‘I’m not going to do it.’”

The ease of enrolling with little more than a credit card may bring to mind diploma mills and doubts about credibility with employers, but that is less a concern for many of the students who seek their education online. “If I grew up in southern rural Indiana, if I say, ‘I have an MBA,’ I’m going to blow people away. They don’t care where that degree came from,” says Phillips. And in a market where objective measures of educational quality are hard to come by, consumers look to price as a signal of quality.

High prices, oddly enough, also keep students enrolled. “If it’s too cheap, the school risks losing its accreditation,” says Eric Bettinger, associate professor of economics and education at Stanford University’s Graduate School of Business. That’s because low prices make it more tempting for students to drop out, and high drop-out rates are a red flag to regional accrediting bodies. Some experts believe that Pell Grants, military subsidies, and other federal student aid, all of which for-profit schools urge students to pursue, have also inflated the price of online schooling.

The Future of Higher Ed

Despite their rapid growth, the for-profits are not giving traditional universities a run for their money just yet. Most students still prefer face-to-face contact and a well-recognized credential. Unlike print newspapers, whose survival the Internet has helped endanger, traditional colleges offer much more than information. The schools that are not secure have more impetus to go online, says GetEducated.com’s Phillips, is “this vast wasteland of private mediocre schools.” She cites schools like Michigan’s Baker College, which has historically catered to the auto industry. With this local

customer base eroding, Baker needed to extend its geographic reach to stay in business. But, as Phillips puts it, “the problem with the Internet is the whole world is your marketplace—and it’s also your competition.”

Specialization can make it easier to compete. By offering classes online, USC’s School of Gerontology, for example, can attract a large number of students even to a niche program for managers of nursing homes. Similarly, Penn State Online offers a master’s degree in homeland security. Indeed, whereas only about 33% of providers of bachelor’s degree programs surveyed by the Sloan Consortium said that online education is critical to their school’s long-term strategy, nearly two-thirds of master’s, doctoral, and specialized programs said so. All that shows the Long Tail is at work in higher education, and, says Hentschke, that tail will only get longer. “The model of the 18- to 22-year-old going to a residential campus like USC and watching football games and that kind of stuff will be around for along time,” he says, “but the demand for other models is coming from lots of other areas and demographics.” ■

Further Reading

Allen, E. and Seaman, J. *Learning on Demand: Online Education in the United States, 2009*. Babson College and The Sloan Consortium, 2009.

Bramble, W.J. and Panda, S.K. *Economics of Distance and Online Learning: Theory, Practice, and Research*. Routledge, New York, NY, 2008.

Carey, K. *College for \$99 a month, Washington Monthly*, September/October 2009.

Kumar, A.N. *The effect of using problem-solving software tutors on the self-confidence of female students*, Proceedings of the Thirty-Ninth Special Interest Group on Computer Science Education, March 12–15, 2008, Portland, OR.

Tierney, W.G. and Hentschke, G.C. *New Players, Different Game: Understanding the Rise of For-profit Colleges and Universities*. Johns Hopkins University Press, Baltimore, MD, 2007.

Based in the San Francisco area, **Marina Krakovsky** is co-author of the forthcoming *Secrets of the Moneylab: How Behavioral Economics Can Improve Your Business*.

© 2010 ACM 0001-0782/10/0900 \$10.00

ACM China Nearing Launch

ACM's expansion into China will support local professionals and increase Chinese involvement in ACM's international activities.

ALTHOUGH CHINA-BASED COMPUTER professionals author numerous papers for international journals and conferences each year, their opportunities are limited by language barriers and a lack of international contacts. At the same time, few outside of China are aware of the variety of computer science developments inside the country's borders. Each year sees thousands of papers published only in Chinese by researchers who are unable to travel to international conferences, so their findings are all but locked off from the international community.

ACM hopes to help change that with the launch of ACM China, whose 20-member Council held its first meeting in June. ACM China's full launch, which is expected later this year, will culminate years of effort to give greater access and exposure to Chinese computer professionals.

China is the latest of three areas outside the U.S. to start an ACM Regional Council in the past year. The

ACM Europe Council launched in October 2009, and the ACM India Council launched in January 2010. But ACM's efforts to grow the organization beyond national borders go back 20 years. "We started encouraging our SIGs [Special Interest Groups] to hold conferences outside the U.S. around 1990, when ACM was viewed as a purely American organization," says ACM Executive Director and Chief Executive Officer John White. "More recently, we've made an effort to have non-U.S. computer scientists in positions of leadership throughout the entire organization."

The selection of Council members was especially critical in China, where leadership reputation is highly valued. ACM China was fortunate to enlist Jianguang Sun, computer science professor and vice president of the National Natural Science Foundation of China, as the Council's chair.

"Once Dr. Sun agreed to help, things progressed very rapidly," recalls Vincent Yun Shen, professor emeritus of the computer science department at

the Hong Kong University of Science and Technology. "That's how Chinese people do business. You have to get the right person to lead a project—someone with credibility. When people learned that Dr. Sun was involved, they said, 'This guy is successful, so aligning with him is a good thing.'"

But establishing ACM China among Chinese computer professionals won't be easy. According to Yunhao Liu, associate professor in the department of computer science at Hong Kong University of Science and Technology, many of them "don't know that IEEE and ACM are different organizations." (Chinese membership in ACM is currently below 2,500.) Liu believes the twin keys to success are effective promotion, and cooperation with the 15,000-member China Computer Federation (CCF).

"CCF has a dominant position among computer societies in China," says Liu, "while few computer scientists here realize that ACM sponsors the famous Turing Award! But CCF's resources are limited when compared to ACM, which is international. ACM is looking forward to close cooperation with CCF and other organizations in China." ACM expects its selections for ACM China's Council to pave the way in building a relationship with CCF, as all but three of them are members of the CCF board or CCF senior members.

With eminent Chinese computer professionals leading the charge, Liu believes the time is right for ACM to enter China. "You have to get people who have been working in China already, and I think Dr. White made the right decisions," Liu says. "That's why I have every confidence that it will be very successful." 

Tom Geller is an Oberlin, OH-based science, technology, and business writer.

© 2010 ACM 0001-0782/10/0900 \$10.00



The ACM China Council meets with ACM COO Pat Ryan and CEO John White (first row, second and fifth from left, respectively) and President Dame Wendy Hall (first row, fourth from left).

Kyoto Prize and Other CS Awards

László Lovász, Vinton G. Cerf, and other researchers are honored for their contributions to computer science.

THE INAMORI FOUNDATION, ACM, and IEEE recently recognized leading computer scientists for their research and service.

Kyoto Prize

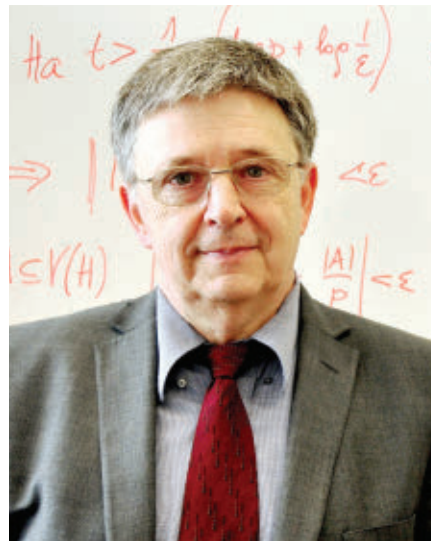
László Lovász, who is director of the Mathematical Institute at Eötvös Loránd University, has been awarded the 26th Annual Kyoto Prize in Basic Sciences from the Inamori Foundation. Lovász is being honored for his outstanding contributions to the advancement of both the academic and technological possibilities of the mathematical sciences.

Lovász has solved several monumental problems, but is perhaps best known for the Lovász local lemma, in which he provides a fundamental probabilistic tool for the analysis of discrete structures, and contributes to the creation of a framework for probabilistically checkable proofs. The basis algorithm, commonly known as the “LLL algorithm,” has also contributed to the construction of important algorithms, and has become a fundamental tool in the theory of cryptography.

ACM Awards

Radia Perlman, an Intel Fellow, has been awarded the highest honor from ACM’s Special Interest Group on Data Communications (SIGCOMM) for pioneering contributions to Internet routing and bridging protocols. SIGCOMM cited Perlman for her work on spanning tree bridging algorithms and link state routing algorithms, advances that have made the Internet more scalable and robust. To this day, both of these algorithms are used in most Internet switching devices.

Christos Faloutsos, a professor at Carnegie Mellon University, received the 2010 Innovation Award from the



Kyoto Prize winner László Lovász

Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) for his contributions to key discoveries in time series database analysis, Internet topology, and Internet auction fraud detection. Faloutsos’ cross-disciplinary works on power-law graphs, fractal-based analysis, time series, multimedia, and spatial indexing are among the most referenced in industry and academic publications. SIGKDD also presented the 2010 Service Award to **Osmar R. Zaiane**, a professor at the University of Alberta, for his dedication to promoting the development of the global KDD community. Zaiane has been furthering data mining through active participation in other industry associations and driving the development of international communities dedicated to the advancement of KDD.

IEEE Awards

IEEE recently paid tribute to leaders in technology at its 2010 Honors Ceremony. Among the awards recipients are:

Vinton G. Cerf, vice president and

chief Internet evangelist for Google, HKN Eminent Members Recognition;

N.R. Narayana Murthy, chairman and chief mentor at Infosys Technologies, Ltd., IEEE Honorary Membership;

Barry Boehm, founding Director Emeritus of the University of Southern California Center for Systems and Software Engineering, Simon Ramo Medal;

John Hopcroft, IBM Professor of Engineering and Applied Mathematics at Cornell University, and **Jeffrey D. Ullman**, Stanford W. Ascherman Professor of Computer Science (Emeritus) at Stanford University, John von Neumann Medal;

Ronald W. Schafer, HP Fellow in the Multimedia Communication and Networking Laboratory at Hewlett-Packard Laboratories, Jack S. Kilby Signal Processing Medal;

Whitfield Diffie, visiting scholar at Stanford University, **Martin E. Hellman**, Professor Emeritus of Electrical Engineering at Stanford University, and **Ralph C. Merkle**, senior research fellow at the Institute for Molecular Manufacturing, Richard W. Hamming Medal;

Randy Howard Katz, United Microelectronics Corporation Distinguished Professor in Electrical Engineering and Computer Science at the University of California, Berkeley, James H. Mulligan, Jr., Education Medal;

Stephen Deering, retired, Internet Award;

Larry Peterson, Robert E. Kahn Professor of Computer Science at Princeton University, Koji Kobayashi Computers and Communications Award;

Toshio Fukuda, professor in the Department of Micro-Nano Systems Engineering at Nagoya University, Robotics and Automation Award. 

Jack Rosenberger is *Communications'* senior editor, news.

© 2010 ACM 0001-0782/10/0900 \$10.00

Introducing:

XRDS

The ACM Magazine for Students

XRDS delivers the tools, resources, knowledge, and connections that computer science students need to succeed in their academic and professional careers!

The All-New **XRDS: Crossroads** is the official magazine for ACM student members featuring:

- › Breaking ideas from top researchers and PhD students
- › Career advice from professors, HR managers, entrepreneurs, and others
- › Interviews and profiles of the biggest names in the field
- › First-hand stories from interns at internationally acclaimed research labs
- › Up-to-date information on the latest conferences, contests, and submission deadlines for grants, scholarships, fellowships, and more!



Also available

The All-New *XRDS.acm.org*

XRDS.acm.org is the new online hub of *XRDS* magazine where you can read the latest news and event announcements, comment on articles, plus share what's happening at your ACM chapter, and more. Get involved by visiting today!

XRDS.acm.org



Association for
Computing Machinery

Advancing Computing as a Science & Profession



DOI:10.1145/1810891.1810902

Phillip G. Armour

The Business of Software Return at Risk

Calculating the likely true cost of projects.

IN A PREVIOUS column,^a I noted that many organizations do not seem to explicitly calculate the “cost of risk” on their projects. Companies may acknowledge risk, identify risk items, implement risk management programs, track risk indicators, and adjust project management actions to mitigate risk. But they often don’t actually compute how much of it there is and what it will likely cost them.

Even businesses such as insurance companies for whom risk quantification is a core competency often fail to assess the cost of the risk they are taking on when they run software projects. This apparently unconscious failure to numerically deal with a critical item of business knowledge seems to extend to other disciplines too. A while ago I came across a financial services company that was routinely performing a quite incorrect calculation of ROI on its software projects.

Straight-line ROI

Imagine a project with an expected cost of (say) \$1 million and an expected return of \$1.1 million. Ignoring issues of

inflation, cost of capital, and alternative investment profits, the ROI for this project appears to be 10%. This project should produce a \$100,000 return on a \$1 million investment. In my experience, this is the most common calculation

performed by companies on most internal development projects. More correctly, it is the calculation done by those companies that actually do estimate their ROI; there are many companies that don’t do it at all or do a very per-



^a P.G. Armour, “Mortality Play,” *Commun. ACM* 50, 3 (Mar. 2007), 15–18.

functionary job of it when they do. But that is a topic for another day. This simple arithmetic could be called a “straight-line ROI.” It is simply the expected returned value divided by the expected cost. It is a simple calculation, easy to compute and to understand. But, in most cases, it is also wrong.

Figure 1. Straight-line ROI.

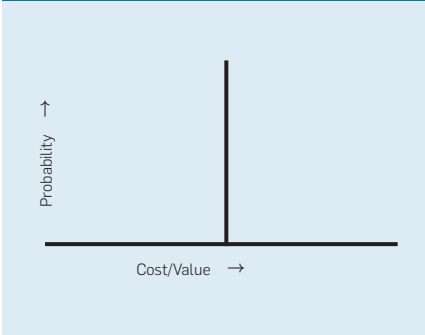


Figure 2. Gaussian probability distribution.

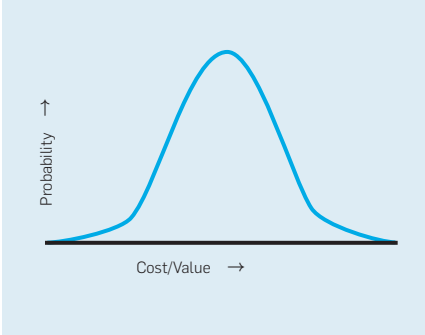


Figure 3. Cost profiles.

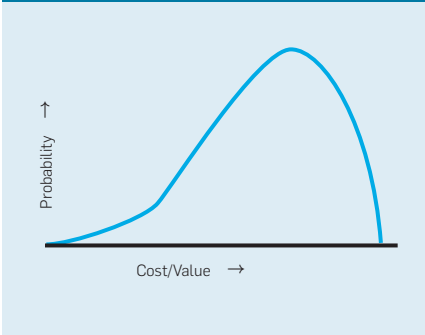
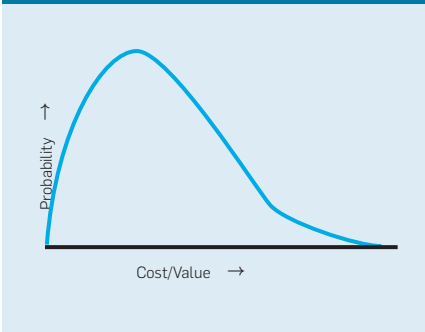


Figure 4. Value profiles.



The Role of Risk

The reason why straight-line ROI is wrong for most projects is simply that it does not account for risk. The return computed using the above straight-line ROI calculation will be true only if there is:

- ▶ 100% guarantee of cost containment at \$1 million—that is, the project has no cost risk, the project cannot/will not run over or under in budget.

- ▶ 100% guarantee of value returned at \$1.1 million—the project has no return risk, the return is fixed and invariable no matter what happens to the project.

These are the necessary conditions for the *calculation* to be valid. There are conditions where the cost risk and value risk cancel out and the calculated return ends up at 10%. For instance, if a project overruns on cost but is able to recoup more value than expected, it may cancel out the budget overrun. Note, this does not mean the calculation is correct, simply that the project was “lucky”^b in that the inaccuracies happened to be equal and opposite.

The moment we introduce risk, the straight-line ROI calculation does not work. If we only have a 20% probability of cost containment at \$1 million, given a typical set of project conditions, the ROI is not a positive 10%, it is more like a negative 18% (!)

Stocks and Bonds

This is true in other disciplines. Equities typically carry more risk than government, municipal, or corporate bonds so we expect higher return to compensate us for the risk. Bonds are safer, so we are content with less income because the downside is more controlled. A less sophisticated financial consultant may show a customer what savings might be achieved over time based on the average returns of the stock market. The U.S. stock market has realized approximately 9% average annual gains over the last 100 years or so (depending on the index used and whether returns are compounded). Showing a potential investor a nice straight line of ever-increas-

ing wealth is more a sales gimmick than a realistic prediction of return since it does not account for risk. Similarly, calculating the likely return for a software project without accounting for risk is bogus.

The financial services company I mentioned would not dream of using a straight-line return calculation for its investments but, like the insurance company that did not calculate cost of risk, it blithely performed the wrong calculation on its internal projects. And it wondered why it got blindsided by failure to achieve returns on most of its projects.

Risk-Weighting

To more correctly calculate the true likely return we must incorporate the cost of risk and its counterpart—what we could call the *value at risk*. Recently, financial markets have shown what failure to properly account for risk does to one’s investment, either through not including risk in the calculation or having the risk hidden inside complex derivatives. It is very important that we learn from this in the business of software by performing the right calculation.

There are six elements to computing return at risk:

- ▶ The expected cost of the project
- ▶ The likelihood of achieving that expected cost
- ▶ The risk profile or “shape” of the cost risk distribution
- ▶ The expected returned value of the project
- ▶ The likelihood of achieving that value
- ▶ The value profile or “shape” of the value distribution

The reason why straight-line ROI is wrong for most projects is simply that it does not account for risk.

^b This is similar to the “lucky” (as opposed to “accurate”) estimate described in P.G. Armour, “Truth and Confidence,” *Crosstalk* (Apr. 2008), 27.

Risk is always paid for somewhere: in the stock market, in insurance underwriting, and in software projects.

Expected Cost/Value

These are typically calculated using traditional estimation approaches. The cost components may be assisted by estimation tools. The value side is usually calculated using some internal assessment of operational cost containment, market expansion, revenue return, and the like. Neither of these processes is particularly easy, but they are quite well-defined.

Likelihood of Cost/Value

These can be computed using techniques such as Monte Carlo analysis operating on the ranges of key variables that contribute to the cost (or value). There are tools available that can perform these calculations easily.^c More sophisticated financial planners will typically use this approach when laying out projections for their customers.

Shape of Risk/Value

This is a complex subject, the detail of which is beyond the scope of this column. The “shape” of risk/value is driven by the expected likelihood of costs and value over- or underrunning. The mistake the straight-line ROI makes it is assumes the risk profile looks like Figure 1. This probability distribution shows there is one and only one likelihood of a result. The chart shows the cost (or delivered value) of the project is 100% guaranteed at the expected value. This means the project is carrying no risk. It also means the project has no unknown factors or variables that affect cost or value. Such projects do not exist in the real world.

The simplest and most common probability distribution is the Gaussian (see Figure 2). With this distribution the likelihood of over- or underrunning budget (or value) from the midpoint “most likely” is equal. While cost and value distributions are rarely symmetrical in real life, this can be a useful distribution provided the “expected” cost or value is set off-center.

Cost profiles usually look something like the curve shown in Figure 3. The distribution shows there is much more likelihood of the project overrunning its budget than there is in underrunning. Unless the likelihood of cost containment is very high (the project’s expected cost is set to the right of the midpoint on the *x*-axis) this project is carrying a high cost of risk.

Value profiles (see Figure 4) are often the reverse of the cost profiles. In general, experience shows we are more likely to underachieve our value goals than we are to overachieve them. Therefore, to better *guarantee* returns we would need to have our expected value moved to the left (we expect lower value delivered) of the midpoint.

Using these models, we can calculate a *risk-weighted* return for our projects and either choose not to carry such a high risk or to more realistically resource our projects based on the challenges they are likely to experience.

Real Return

The difference between straight-line return and risk-weighted return is simply the aggregate cost of risk as expressed in the likelihood of a project both running over budget and underachieving in the value it delivers. Risk is always paid for somewhere: in the stock market, in insurance underwriting, and in software projects. It seems that few companies perform this kind of calculation, even when it is one of their core competencies—which is odd to say the least. In the business of software, we can’t complain about our performance if we resource our projects but don’t quantify and resource the *risk* on the projects. When that risk comes home—and it will—our projects will fail. And they do. ■

Phillip G. Armour (armour@corvusintl.com) is a senior consultant at Corvus International Inc., Deer Park, IL.

Copyright held by author.

Calendar of Events

September 15–17

Audio Mostly,
Pitea, Sweden,
Contact: Delsing Katarina,
Email: katarinea.delsing@tii.se

September 15–17

Principles and Practice of
Programming in Java,
Vienna, Austria,
Contact: Krall Andreas,
Email: andi@complang.
tuweien.ac.at

September 15–18

Quantitative Evaluation of
Systems,
Williamsburg, VA,
Contact: Evgenia Smirni,
Email: esmirn@cs.wm.edu

September 16–17

2010 ACM/IEEE International
Symposium on Empirical
Software Engineering and
Measurement,
Bolzano, Italy,
Contact: Giancarlo Succi,
Email: Giancarlo.succi@unibiz.it

September 16–17

HCI in Work & Learning, Life &
Leisure,
Carinthia, Austria,
Contact: Herbert Janing,
Email: aapc2010@uni-klu.ac.at

September 20–24

IEEE/ACM International
Conference on Automated
Software Engineering,
Antwerp, Belgium,
Sponsored: SIGSOFT, SIGART,
Contact: Charles Pecheur,
Email: charles.pecheur@
uclouvain.be

September 20–24

The 16th Annual International
Conference on Mobile
Computing and Networking
and the 11th ACM International
Symposium on Mobile Ad Hoc
Networking and Computing,
Chicago, IL,
Sponsored: SIGMOBILE,
Contact: Nitin H. Vaidya,
Email: nhv@uiuc.edu

September 21–22

International Symposium on
Intelligent Virtual Agents,
Philadelphia, PA,
Contact: Norman I Badler,
Email: badler@seas.upenn.edu

^c For example, the The SLIM-Estimate tool, marketed by QSM Inc., McLean, VA, can quote explicit cost of risk.

Law and Technology

Principles of the Law of Software Contracts

An overview of a new set of legal principles for software contracts developed by the American Law Institute.

IN THE UNITED STATES as in many countries, the software industry is increasingly important. Proprietary and open source software powers items as diverse as PCs and refrigerators, and controls systems as vital as missile defense and the utility grid. Many software vendors' principle source of revenue comes from licensing their code to businesses and/or consumers. Others use software as a means to drive demand for another money-making product like services. In any event, no one seriously questions software's place in the economy or its importance to modern life. It may be somewhat surprising then that the law of software transactions in the U.S. has not been uniform. In Europe, European Union directives,^a such as Council Directive 2009/24/EC of 23 April 2009 on the legal protection of computer programs (EU Directive) (replacing the former directive from 1991), helpfully set forth basic principles. In the U.S., courts look to the common law of contract (that is, the decisions of courts) or the Uniform Commercial Code (UCC) (a statute enacted by each state) for the rule of decision, and must also consider other sources such as consumer protection law and federal intellectual property law. Contract law and consumer protection law can vary by state and

Software's unique attributes and importance to the economy make legal uniformity and clarity particularly important.

interpretations of the UCC and federal law can also be inconsistent. Although this variation characterizes the U.S. approach to much of its law, software's unique attributes and importance to the economy make legal uniformity and clarity particularly important.

Against this background, the American Law Institute (ALI), a law-reform organization in the U.S.,^b undertook a project to analyze the area of software contracting and to set forth principles that a court could adopt as the rule of decision in a case before it. This effort produced the *Principles of the Law of Software Contracts*, a volume that we drafted and that underwent extensive review over a five-year period. The Prin-

ciples address a set of topics important to software transactions—some unique to the software context, others not. We discuss briefly here just a few of the more important or controversial provisions and recommend that interested readers refer to the complete work for more information.

Scope

The first question the Principles faced was how to define the transactions to which they applied. The Principles intentionally define their scope narrowly to software as traditionally understood. As drafters, we understood the danger of over-inclusiveness: In particular, we wished to avoid the trap of including all types of digital information within our project's scope. As a result, digital media and digital databases are not part of the project.

Even with this narrow approach, questions remained. The Free Software Foundation, at least until the release of Version 3.0 of the General Public License (GPL), maintained that open source licenses akin to the GPL were not contracts under U.S. law, but rather were mere copyright permissions. We disagreed, arguing that under U.S. law as traditionally understood, most open source licenses are indeed contracts because they are in the nature of an exchange between the provider and user. The Principles therefore apply to open source agreements with exceptions and specific provisions where necessary.

^a EU directives consist of legislation that direct member states to promulgate rules to realize a particular goal. See Wikipedia, available at [http://en.wikipedia.org/wiki/Directive_\(European_Union\)](http://en.wikipedia.org/wiki/Directive_(European_Union))

^b For information about the ALI, see, see <http://www.ali.org/index.cfm?fuseaction=about.overview>



Relationship to Intellectual Property Law, Public Policy, and Unconscionability

In the U.S., software agreements as consensual transactions implicate contract law, which is primarily the province of the states. Software, of course, can be protected by federal intellectual property rights, including copyright and patent. Questions can arise whether a state court may enforce provisions under its contract law that provide greater rights or restrict limitations that federal intellectual property law would grant in the absence of the parties' agreement.

Here, as in many areas, the Prin-

ciples take the general position that parties are free to contract as they see fit. The Principles, however, note that courts must be particularly attentive to provisions affecting federal intellectual property rights in the case of boilerplate standard forms. Especially because of the take-it-or-leave-it nature of such forms and the tendency of consumers and others to fail to read them, the federal interest in state non-interference with the intellectual property system is at its height.

For example, many boilerplate agreements include a provision against reverse engineering. Under both the Uniform Computer Information Transactions Act (UCITA) and the EU Directive, such provisions are unenforceable in certain circumstances.^c However, the Principles opt to direct courts to

consider all facts and circumstances, including whether the ban on reverse engineering is in a standard form, rather than adopting a blanket rule.

Many legal doctrines in the U.S. take a similar contextual approach. Some police contractual provisions for fairness in their formation and substance or for their effect on third parties. In this regard, the Principles include sections on unconscionability and public policy. Here again, the Principles do not set forth a list of suspect or unenforceable terms. Instead, they take the traditional U.S. approach of considering the context. But the Principles provide extensive comments about the nature of reasonable contract-formation processes and the fairness of substantive terms to guide courts in their contextual approach.

The Principles opt to direct courts to consider all facts and circumstances, rather than adopting a blanket rule.

^c UCITA was an attempt by a private U.S. organization, the National Conference of Commissioners on Uniform State Law, to promulgate a uniform law governing "information" transactions. It met with much opposition, in part because of its perceived business orientation, and only two states have adopted it. Generally UCITA and the EU Directive preclude enforcement of a provision against reverse engineering if necessary to obtain information for interoperability purposes.

Implied Warranty of No Material Hidden Defects

The Principles clarify warranty law that has become muddled particularly under the UCC. For example, in the Principles, the creation of an express warranty and whether it is disclaimed depend respectively on whether a reasonable person could rely on the rep-

resentation and whether a reasonable person would be surprised by the disclaimer. One warranty provision, however, has been controversial. Section 3.05(b) provides that a party who transfers software and receives money or a monetary obligation in return warrants “that the software contains no material hidden defects of which the transferor was aware at the time of the transfer.”

Software providers objected to this provision as inconsistent with current law and likely to increase litigation. They also believe the warranty should be disclaimable. However, the rule merely codifies U.S. contract law’s duty to disclose and obligation of good faith and tort law’s fraudulent concealment principle. Further, the material-hidden-defect rule should not be difficult to administer. It defines a material defect as one that constitutes a material breach of the agreement. As such, the rule draws on the well-rehearsed material breach doctrine of U.S. law. Additionally, a hidden defect is one the provider knows about but would not surface upon any testing that was or should have been performed by the user. Disclosure of the defect occurs when a reasonable user would understand the existence and nature of the defect. As the Principles point out, providers that do not engage in concealment should have little to fear from this rule. But contract law should not support a provider’s strategy to foist a product known to be materially defective on to a user without providing that user with a remedy for potentially significant losses. Providers can insulate themselves from liability by disclosing material defects in their software.

Automated Disablement

Another section that exposed conflicting views governs automated disablement. Automated disablement refers to a provider’s use of electronic means to disable or materially impair the functionality of software such as by building in a “time bomb” or accessing the user’s system remotely to disable certain software.^d The Principles severely limit

^d If a user accesses a provider’s software by connecting to software resident on the provider’s system, failure of the provider to grant access is not considered an automated disablement.

It is better to provide protections to all firms rather than trying to distinguish between those that are knowledgeable and informed and those that are not.

the use of automated disablement as a remedy for breach: It is unavailable at all in the case of a consumer transaction or a standard-form transfer of generally available software. Additionally, the term authorizing automated disablement must appear conspicuously in the agreement, the party seeking to employ automated disablement must provide notice and an opportunity to cure to the user, and the provider must obtain a court order before disabling the software. These obligations are not disclaimable and damages for their breach may not be limited.

Particularly those software providers marketing to large, knowledgeable, well-informed commercial parties objected to the restrictions placed on automated disablement as too onerous and an unwarranted intrusion on freedom of contract. They also objected to the non-disclaimable nature of the obligations and inability to limit damages.

The commentary to the automated disablement section recognizes both these concerns and the historical nature of the debate. Software providers argued that automated disablement is necessary to prevent ongoing misuse of the software or a continuing breach that is causing damages to accumulate without any real possibility the provider will ever be made whole. Users argue that breach is highly contextual and a wrongful denial of use may cripple a business and/or harm software not even implicated in the dispute. Moreover, allowing providers to leave a “back door” open to permit automated disablement pos-

es real security risks. Automated disablement is so controversial that, as recently as 2002, UCITA prohibited its use.

We believe our approach to automated disablement presents a reasonable balance between the conflicting interests. Even when commercial entities negotiate contracts, one side may overreach. Firms using software are not monolithic—many are small firms that cannot afford to hire lawyers to negotiate complex provisions. These firms are more akin to consumers than to large businesses. It is better to provide protections to all firms rather than trying to distinguish between those that are knowledgeable and informed and those that are not.

What’s Next?

The Principles, of course, contain many other provisions, including standards for enforcement of online contracts, which we do not have space to discuss here. Rather, we would simply emphasize a few points:

- The Principles were the result of a five-year drafting process that included input from both users and providers of software. There were many contentious issues on which both constituencies would never agree. In such cases, we made difficult choices informed by the legitimate points raised by both sides. We are not surprised the result sometimes makes neither side happy.

- The Principles are not the law of the U.S. or any particular state in the U.S. One or more of their provisions would become law if a court in a concrete case chose to adopt them as its rule of decision.

- The Principles address both topics that are unique to software and those that are not. Their applicability is limited to their scope. To the extent, however, that courts or commentators find their approach useful outside of the software context, we would welcome their use by analogy in other areas. C

Robert A. Hillman (rah16@cornell.edu) is the Edward H. Woodruff Professor of Law at Cornell Law School, Ithaca, NY, and was the Reporter of the Principles of the Law of Software Contracts.

Maureen A. O’Rourke (morourke@bu.edu) is Dean, Professor of Law, and Michaels Faculty Research Scholar at Boston University School of Law, Boston, MA, and was the Associate Reporter of the Principles.

Copyright held by author.

The Profession of IT

Discussing Cyber Attack

Cyber attack—the other side of cyber defense—deserves a more open discussion than it has been getting.

DEFEND OUR NETWORKS!" is the new rallying cry in a time of rising concerns over cyber vulnerabilities. Malware, Trojan horses, computer system weaknesses, network vulnerabilities, intrusions, data theft, identity theft, malicious botnets, and critical infrastructure protection are under constant discussion. Computing professionals are called on daily to help with these problems. Cyber defense is the topic of hundreds of conferences and research papers every year.

By contrast, cyber attack, the flip side of defense, has been a touchy subject. Many people feel uneasy when they hear their governments want to be in a position to launch cyber attacks. Most public discussions of cyber attack tend to focus on the "bad guys" (unauthorized individuals with malicious intent) who launch the attacks and the methods they use—all for the purpose of developing better defenses. Governments are quiet about not only their cyber attack methods and operations, but also the policies they follow. This secretiveness has fueled many fears that governments are up to things the citizens would disapprove.

Yet there is a growing international public discussion on cyber attack, promoted in part by reports of government activity in the area. The U.S. Department of Defense established the U.S. Cyber Command earlier this year to coordinate the cyber defense of military networks and to direct military cyber



Defense Secretary Robert Gates addresses the audience with Gen. Kevin Chilton, commander, U.S. Strategic Command, and Gen. Keith Alexander, commander, U.S. Cyber Command, during the activation ceremony of U.S. Cyber Command on Fort Meade, MD, May 21, 2010.

attacks. Other militaries are doing the same. Security experts Richard Clarke and Robert Knake believe that cyber attacks and cyber war are already under way.¹ Massive denial-of-service attacks against government sites in Estonia in 2001 and Georgia in 2008 led to charges that Russia was engaging in cyber warfare. China was blamed for infiltrating and stealing sensitive data from Google's network and other targets in 2009. Many believe that cyber espionage by government intelligence agencies is widespread.

There is an important role for computer professionals in the discussions and other activities in this area. To

point the direction, we will use a recent report on cyber attack from the National Research Council.³ The report, which addresses the technical, policy, legal, and ethical dimensions of cyber attack, makes important distinctions that are useful to frame the discussion. While written for the U.S., it discusses the issue in a way that relates to many countries.

Cyber Attack and Exploitation

Cyber attack refers to deliberate actions against data, software, or hardware in computer systems or networks. The actions may destroy, disrupt, degrade, or deny access.



ACM's *interactions* magazine explores critical relationships between experiences, people, and technology, showcasing emerging innovations and industry leaders from around the world across important applications of design thinking and the broadening field of the interaction design. Our readers represent a growing community of practice that is of increasing and vital global importance.

interactions
<http://www.acm.org/subscribe>



Computing technologies open many options and complexities that more casual users do not appreciate.

Many governments' militaries and intelligence agencies are actively preparing to engage in cyber attacks, perhaps in conjunction with conventional attacks or counterattacks.

Cyber exploitation is another term in the discussions. It refers to intelligence-gathering rather than destructive activities. Cyber exploitation usually seeks the least intrusive, least detectable interventions into computing systems. The purpose is to acquire data without being seen or getting caught. Exploitation also refers to forensic recovery of data from discarded (or captured) laptops and storage media.

Both attack and exploitation require three things: access to a system or network, vulnerabilities in the accessed systems, and a payload. The access might be remote through the Internet or close-in through physical access. Vulnerabilities can appear in hardware, software, hardware-software interfaces, communication channels, configuration tables, users, and service providers. The payload is a program that performs actions once a vulnerability has been found and exercised. A payload might be a bot, data monitoring program, virus, worm, spyware, or Trojan horse; and it is likely to have remote access to the attacker's communication channels. The difference between attack and exploitation depends on the actions of the payload. An attack payload is destructive, an exploit payload is nondestructive. Often the differences are so subtle that the victim of a cyber operation may not be able to tell as it is happening which it is.

Cyber attack and exploitation are tools used in the service of larger ends. They offer a new range of capabilities

to government that can be more humane and less collaterally damaging than their traditional "kinetic" predecessors. For example, a military operation may depend on disabling an adversary's radars scattered around a city; if a cyber attack could disable the radars, there would be no need to bomb the installations and suffer all the collateral damage those bombings would entail. An intelligence operation that can steal files remotely avoids risking the lives of its secret agents. However, people who would accept these ends might also worry about the same tools being used for other ends, such as a government agency spying on its citizens.

The NRC report discusses the technical, policy, and social aspects of cyber attack and exploit. It identifies complicated issues that must be resolved in such areas as the law of armed conflict, deterrence, and the dynamics of cyber attack. While the principles underlying the United Nations charter on the use of force and armed attack offer a good starting point for an international regime governing cyber attacks, they are difficult to apply to many cyber attacks. Traditional policies of deterrence by threat of overwhelming response are problematic in cyberspace because of the extreme difficulty of accurately identifying perpetrators. The dynamics of cyber attack are also poorly understood, including how to keep a cyber conflict from escalating out of control and how to terminate cyber conflict. The report recommends that these and other issues be discussed in an open, public debate.

The Need for Technical Expertise

It's tempting for us to say that these issues look primarily legal, ethical, or political, and that we should let lawyers, ethicists, and politicians look after them. That reasoning is unsound. Computing technologies open many options and complexities that more casual users do not appreciate. Computing professional advice on the capabilities and limits of the technology is crucial to the formulation of sound policies, as well as the development of tools for attack, exploit, and defense.

A significant example of this occurred in 1985 when the U.S. government undertook the Strategic Defense Initiative (SDI), an automated missile

defense system. Many computing people initially declined to join the debate because they believed it was inherently political and they had little to offer. That changed with David Parnas's remarkable *Communications* article, "Software aspects of strategic defense systems,"⁴ which set out for the first time the scientific framework of software engineering. Parnas showed that software engineering at the time was not capable of producing reliable control systems for missile defense. After that many computing professionals joined the debate to add their own experience and expertise with unreliable large, complex systems.

There are several other examples where political and legal issues depended on an understanding of the limits of computing technology, and computing professionals made important contributions to the debates. These included the move toward e-voting, cryptography policy, architecting the Internet for strong authentication, technologies to improve or impede anonymity, proposals to charge postage on email to stop spam, and network neutrality.

Cyber attack is on par with the strategic defense issue. The complex and subtle issues of cyber attack cannot be adequately resolved unless experts knowledgeable in the workings and capabilities of information technologies participate actively in the discussions. Some of the areas where technical expertise is essential include:

- Advancing the capabilities for rapid attribution—determining who instigated an attack so as to enable a timely and precise response.
- Understanding and measuring

both direct and indirect effects of cyber attacks; assessing damages related to direct and indirect effects of cyber attacks.

- Determining whether a cyber operation is an attack or exploitation—or generally inferring intent.

► Trying to understand, through war game simulations, how social and technical systems in the Internet might respond to various attacks and provocations, how cyber attacks could escalate out of control, and which "games of cooperation" might best thwart attacks.

► Understanding the relationship between recovery time and value of an attack—an attacker is less motivated to take down a network if the victim can quickly restore it to operation.

► Finding effective means of planting or discovering Trojan horses and other forms of malware.

► Determining the effects of virtualization in the cloud on the ability to mount, detect, and thwart attacks.

► Understanding and minimizing risks introduced by development or use of cyber attack and exploit capabilities.

► Understanding and explaining implications of new technologies—how they might be attacked or how they might facilitate an attack or exploit. For example, technologies for smart grids, smart cars, wireless home networks, or social networking systems.

► Determining the requirements for getting good indications and warnings of cyber attack—is it necessary to penetrate adversary networks to get this in a timely enough manner to defend or respond effectively?

Studying these areas contributes to better defenses. It is not possible to build strong defenses without acquiring and maintaining a solid understanding of how attacks work and how effective they might be.

What You Can Do

It is important that computing professionals bring their general knowledge of computers and networks to the discussions of technical, policy, legal, and social issues around cyber attack. There are several ways to do this:

- Engaging in research in the above areas and publishing results.
- Developing and participating in cyber attack and defense exercises; mak-

ing sure that cyber exercises are true to technology and its limits.

► Participating in groups that address cyber attack issues, for example, the Cyber Conflict Studies Association (cyberconflict.org), which sponsors meetings and working groups on various topics relating to cyber attack and defense.

► Participating in online discussion groups such as the Cyber Security Forum Initiative's Cyber Warfare Division (CSFI-CWD) on LinkedIn.

► Participating in conferences such as InfoWarCon (cyberloop.org) or the Conference on Cyber Conflict sponsored by the NATO-accredited Cooperative Cyber Defence Centre of Excellence in Estonia (www.ccdcoe.org).

► Participating in government-sponsored working groups that address cyber attack issues.

► Separating truth from fiction about technology in media stories—writing articles that debunk myths.

Even though many of the meetings and discussions on cyber conflict emphasize the legal and policy issues, it is vital that computing professionals participate so that findings and recommendations are based on a sound understanding of technology. Moreover, the networks of computing professionals formed in these discussions become powerful resources for responding to cyber attacks.

We join with the NRC report to strongly endorse the strategy of openness in these efforts and discussions. Openness mobilizes many brains on difficult problems, increasing the chances of finding good solutions. ■

References

1. Clarke, R., and R. Knake. *Cyber War*. Ecco, 2010.
2. Denning, D. E. *Information Warfare and Security*. Addison-Wesley, 1998.
3. National Research Council. *Technology, Policy, Law, and Ethics Regarding U. S. Acquisition and Use of Cyberattack Capabilities*. W.A. Owens, K.W. Dam, and H.S. Lin, Eds., National Academic Press, 2009. Available from MacArthur Foundation, macfound.org, search for "cyberattack."
4. Parnas, D. Software aspects of strategic defense systems. *Commun. ACM* 28, 12 (Dec. 1985), 1326–1335.
5. Vijayan, J. Over 75,000 systems compromised in cyberattack. *Computerworld* (Feb 18, 2010).

Peter J. Denning (pjd@nps.edu) is Distinguished Professor of Computer Science and Director of the Cebrowski Institute for Innovation and Information Superiority at the Naval Postgraduate School in Monterey, CA and is a past president of ACM.

Dorothy E. Denning (dedennin@nps.edu) is Distinguished Professor of Defense Analysis at the Naval Postgraduate School in Monterey, CA, and author of *Information Warfare and Security*.²

Copyright held by author.

It is not possible to build strong defenses without acquiring a solid understanding of how attacks work and how effective they might be.

Viewpoint

Objects Never? Well, Hardly Ever!

Revisiting the Great Objects Debate.

AT THE 2005 SIGCSE (Special Interest Group in Computer Science Education) Symposium in St. Louis, MO, a packed audience listened to the Great Objects Debate: Should we teach “objects first” or “objects later”?¹ In the objects-first approach, novices are taught object-oriented programming (OOP) in their initial introduction to programming, as opposed to an objects-later approach, where novices are first introduced to procedural programming, leaving OOP to the end of the first semester or the end of the first year. Kim Bruce and Michael Kölling spoke in favor of the objects-first approach, while their opponents Stuart Reges and Eliot Koffman argued for teaching procedural programming first. One of Bruce’s arguments was: since OOP is dominant in the world of software development, it should be taught early. I later contacted Bruce to ask for a warrant for the dominance of OOP, but he could not give me one, nor could any of several other experts to whom I posed the same question.

I claim that the use of OOP is not as prevalent as most people believe, that it is not as successful as its proponents claim, and, therefore, that its central place in the CS curriculum is not justified.

Is OOP Dominant?

In assessing the dominance of OOP, we have to watch out for proxies. The extensive use of languages that support OOP proves nothing, because languages are

chosen for a myriad of reasons, not necessarily for their suitability for OOP, nor for the suitability of OOP itself. Similarly, the use of a CASE tool that supports OOP is another proxy; these tools might just be convenient and effective for expressing the software design of a system, whether OOP is being used or not. Furthermore, many practices associated with OOP, such as decomposing software into modules and separating the interface from the implementation, are not limited to OOP; they are simply good software practice and have been supported by modern programming languages and systems for years.

The classical definition of OOP was given by Peter Wegner⁹: object-oriented = objects + classes

+ inheritance. The Java Swing GUI library, which makes massive use of inheritance, is frequently mentioned as a successful example of software that was designed using object-orientation and it certainly fits Wegner’s definition. Is this style of programming truly dominant?

There is a claim that 90% of all code is being written for embedded systems.⁷ I could not locate the author’s source, but it doesn’t really matter since the claim is just as suspect as the claim that OOP is dominant. However, embedded system development is surely an important field of software, and, based upon my experience, I do not believe that OOP has a significant contribution to make here



because the main challenges are not in the software design itself. The challenges arise from an “unfriendly environment”: getting proprietary hardware to work, obtaining meaningful requirements while the system itself is being designed, integration with non-standard networks and busses, and, above all, determining out how to test and verify the software.

Consider another field where the dominance of OOP is questionable. The development and implementation of new algorithms form the heart of many applications areas like numerical simulation (for example, climate modeling) and image processing (for example of satellite imagery). The challenges arise from mathematical difficulties and demands for performance, and OOP has little to contribute to meeting these challenges.

Not only is there no evidence to back up the claims for the dominance of OOP, but there is criticism of OOP, some of it quite harsh.^{3,8} I, too, have found OOP to be extremely disappointing and I will explain my position from a personal perspective.

What the “Real World” is Really Like

Suppose you ask your students to design OOP software for a car; you would probably give a good grade for the example shown in Figure 1. The only problem is that the real world doesn’t work this way. A wonderful image in a paper by Grimm⁵ shows a schematic diagram for the computer system of the Mercedes-Benz S-class car. The legend for the schematic diagram indicates there are over 50 controllers, 600,000 lines of code, hundreds of bus messages, thousands of signals, and three networks. The details of this system are proprietary, but I am confident that no one sat down and used OOP to “design the software,” for example, by deriving classes as shown in Figure 1. Almost certainly, the vari-

Isn’t it just possible that my inability to profit from OOP reflects a problem with OOP itself and not my own incompetence?

ous subsystems were subcontracted to different companies who jealously guard their software because they are engaged in merciless competition.

The interface to the brake system will be implemented by network protocols and bus signals, and the commands to the brakes will be given as bits and bytes (or even by a hardware specification like “apply the brakes when lines 1 and 5 are asserted continuously for at least 10 milliseconds”). An abstract specification like `void ApplyBrakes()` is meaningless here. More importantly, what is likely to be changed is the *interface*, contrary to the OOP approach, which assumes that different implementations will be “swapped” at a single interface. Let us imagine that at some time in the future the brake manufacturer is asked to supply systems to Daimler competitor BMW. The mechanics, hydraulics, electronics, and algorithms will be reused, but the network protocols and bus signals will certainly require significant modification to fit the systems architecture used by BMW.

I believe that industrial systems are successful because the decomposition is not into classes, but into subsystems. The Mercedes-Benz car has, on the average, $600,000/50 = 12,000$ source code lines per controller, so each individual

subsystem can be developed by a relatively small team in a relatively short time. There is a need for talented systems engineers to specify and integrate the subsystems, but there is no overall grand software design where OOP might help.

Natural and Intuitive

In the 43 years since I first learned to program, I have frequently become excited about developments in programming, such as pattern matching (which I first encountered in SNOBOL) and strong type checking (a revelation when I first learned Pascal), and I found that these new constructs naturally and intuitively supported solutions to programming tasks. I have never had the same feeling about OOP, despite teaching it, writing textbooks on OOP languages, and developing pedagogical software in Java. During all this time, I found only one natural use of inheritance. (I developed a tool for learning distributed algorithms² and found it convenient to declare an abstract class containing the common fields and methods of the algorithms and then to declare derived classes for specific algorithms.) Isn’t it just possible that my inability to profit from OOP reflects a problem with OOP itself and not my own incompetence?

I am not the only one whose intuition fails when it comes to OOP. Har- dar and Leron recently investigated the acquisition of OOP concepts by experienced software developers. They found that: “Under the demands of abstraction, formalization, and executability, the formal OO paradigm has come to sometimes clash with the very intuitions that produced it.”⁶ Again, isn’t it just possible that the intuition of experienced software engineers is perfectly OK, and that it is OOP that is not intuitive and frequently even artificial?

Reuse from the Trenches

One of the strongest claims in favor of OOP is that it facilitates reuse. I would like to see evidence to support this, because, in my experience, OOP makes reuse difficult, if not impossible. Here, I would like to describe two attempts at reuse where I truly felt that OOP was the problem and not the solution. I would like to emphasize that—as far as I can judge—these programs were

Figure 1. Example OOP software for a car.

```
abstract class Brake {
    public abstract void applyBrakes();
}
class DiskBrake extends Brake { ... }
class DrumBrake extends Brake { ... }
```

ACM's Online Books & Courses Programs!

Helping Members Meet Today's Career Challenges

Over 3,500 Online Courses and 1,000 Virtual Labs from Element K!

ACM's Online Course Collection includes over **3,500 online courses, 1,000 virtual labs, e-reference tools and offline capability**. Program highlights:

The ACM E-Learning Catalog - round-the-clock access to 3,500 online courses on a wide range of computing and business topics, in multiple languages.

Exclusive vLab® Virtual Labs - 1,000 unique vLab® exercises place users on systems using real hardware and software allowing them to gain important job-related experience.

Reference Tools - an e-Reference Library extends technical knowledge outside of the classroom, plus online Executive Summaries and quick reference cards to answer on-the-job questions instantly.

Offline Player - members can access assessments and self-study courses offline, anywhere and anytime, without a live Internet connection. A downloadable Quick Reference Guide and a 15-minute site orientation course for new users are also available to help members get started.

The ACM Online Course Program is open to ACM Professional and Student Members.



600 Online Books from Safari

ACM members are eligible for a **special 40% savings** offer to upgrade to a Premium or Full Library subscription. For more details visit: pd.acm.org



The ACM Online Books Collection includes **full access to 600 online books** from Safari® Books Online, featuring leading publishers including O'Reilly. Safari puts a complete IT and business e-reference library right on your desktop. Available to ACM Professional Members, Safari will help you zero in on exactly the information you need, right when you need it.



Association for
Computing Machinery

Advancing Computing as a Science & Profession

500 Online Books from Books24x7

All Professional and Student Members also have **full access to 500 online books** from Books24x7®, in ACM's rotating collection of complete unabridged books on the hottest computing topics. This virtual library puts information at your fingertips. Search, bookmark, or read cover-to-cover. Your bookshelf allows for quick retrieval and bookmarks let you easily return to specific places in a book.



pd.acm.org

www.acm.org/join

designed according to the principles of OOP, and the quality of the design and programming was excellent.

I developed the first concurrency simulator for teaching based upon a Pascal interpreter written by Niklaus Wirth. Several years ago, while looking for a modern concurrency simulator, I found a third-generation descendant of my simulator: an interpreter written in Java, extended with a debugger that had a Swing-based GUI. I wished to modify this software to interpret additional byte codes and to expand the GUI by including an editor and a command to invoke the compiler.

The heart of an interpreter is a large switch/case-statement on the instruction codes. An often-cited advantage of OOP is its ability to replace these statements with dynamic dispatching. In the Java program, an abstract class for byte codes was defined, and from it, other abstract and concrete classes were derived for each of the byte codes. I simply found it more difficult (even with Eclipse) to browse and modify 80 classes than I did when there were 80 alternatives of a case-statement in Pascal.

This was only an annoyance; the real problem quickly surfaced. The extreme encapsulation encouraged by OOP caused untold complexity, because objects have to be passed to various classes via constructors. For example, in the original program, when a button is clicked to request the display of the history window, the statement performed in the event handler is as shown in Figure 2. Well, the history window is derived from an abstract window class, so OOP makes sense here, but there is one debugger, one debugger frame, one interpreter, and one window manager. Why can't these subsystems be declared publicly (and implemented privately) without the baggage of allocated objects and constructors? My attempt to modify the software was continually plagued by the need to access one of these subsystems from a class that had not been passed the proper object. This resulted in cascades of modifications and complicated the task considerably; in addition, it led to a decline in coherence and cohesion. As a result of this experience, I have ceased to automatically encapsulate everything; instead,

Figure 2. Example statement performed in the event handler.

```

getDebugger().getDebuggerFrame().getWindowManager().
    showHistoryWindow(
        getDebugger(), getDebugger().getInterpreter());

```

I judge each case on its own merits. In general, I see nothing wrong with declaring record types and subsystem objects publicly, encapsulating only the implementation of data structures that are likely to change.

My second attempt at reusing OOP software involved a software tool VN that I developed for learning nondeterminism. It takes as input the XML description of a nondeterministic finite automaton that is generated by an interactive graphical tool for studying automata. To facilitate using VN as a single program, I decided to extract the graphics editor from the other tool. But OOP is about classes and Java enables the use of any public declaration anywhere in a program just by giving its fully expanded name. There were just enough such references to induce a cascade of dependencies when I tried to extract the Java package containing the graphics editor.

This is precisely the issue I raised with the imaginary brake system. What I wanted to reuse was the implementation of the graphics editor even if that meant modifying the interface. I saw that I would have had to study many of the 400 or so classes in 40 packages, just to extract one package. The effort did not seem worthwhile, so I gave up the idea of reusing the package and

included the (very large) jar file of the other tool in my distribution.

Paradigms

I suspect I know what your next question is going to be: What paradigm do you propose instead of OOP? Ever heretical, I would like to question the whole concept of programming paradigm. What paradigms are used to design bridges? My guess is the concept of paradigm does not exist there. Engineering design is done by using technology to implement requirements. The engineer starts from data (length of the bridge, depth of the water, characteristics of the river bed) and constraints (budget, schedule), and she has technology to use in her design: architecture (cables, stays, trusses) and materials (steel, concrete). I simply don't see a set of alternative "paradigms" for building bridges.

Similarly, the software engineer is faced with requirements and constraints, and is required to meet them with technology: computer architectures, communication links, operating systems, programming languages, libraries, and so on. Systems are constructed in complex ways from these technologies, and the concept of programming paradigm is of little use in the real world.

Hegemony

It is easy (and not incorrect) to dismiss what I have written as personal opinion and anecdotes, just as I have dismissed OOP as based upon personal opinion and anecdotes without solid evidence to support its claims. But the difference between me and the proponents of OOP is that I am not making any hegemonic claims for my opinions. I do not believe there is a "most successful" way of structuring software nor that any method is "dominant." This hegemony is particularly apparent in CS education, as evidenced by the objects-first vs. objects-later debate concerning teaching OOP to novices. No one questions whether

OOP is at all appropriate for novices, and no one suggests an objects-as-an-upper-level-elective approach or an objects-in-graduate-school approach. Perhaps the time has come to do so.

Conclusion

I will conclude with a "to-do list":

- Proponents of OOP should publish analyses of successes and failures of OOP, and use these to clearly and explicitly characterize the domains in which OOP can be recommended.

- Software engineers should always use their judgment when choosing tools and techniques and not be carried away by unsubstantiated claims. Even if you are constrained to use a language or tool that supports OOP, that in itself is not a reason to use OOP as a design method if you judge it is as not appropriate.

- Educators should ensure students are given a broad exposure to programming languages and techniques. I would especially like to see the education of novices become more diverse. No harm will come to them if they see objects very, very, late. 

References

1. Astrachan, O., Bruce, K., Koffman, E., Kölling, M., and Reges, S. Resolved: Objects early has failed. *SIGCSE Bulletin* 37, 1 (Feb. 2005), 451–452. DOI: <http://doi.acm.org/10.1145/1047124.1047359>.
2. Ben-Ari, M. Distributed algorithms in Java. *SIGCSE Bulletin* 29, 3 (Sept. 1997), 62–64. DOI: <http://doi.acm.org/10.1145/268809.268840>.
3. Gabriel, R. *Objects Have Failed: Notes for a Debate*, (2002); <http://www.dreamsongs.com/Files/ObjectsHaveFailed.pdf>.
4. Gries, D. A principled approach to teaching OOP first. *SIGCSE Bulletin* 40, 1 (Feb. 2008), 31–35. DOI: <http://doi.acm.org/10.1145/1352322.1352149>.
5. Grimm, K. Software technology in an automotive company: Major challenges. In *Proceedings of the 25th international Conference on Software Engineering* (Portland, OR, May 3–10, 2003). International Conference on Software Engineering. IEEE Computer Society, Washington, D.C., 498–503.
6. Hadar, I. and Leron, U. How intuitive is object-oriented design? *Commun. ACM* 51, 5 (May 2008), 41–46. DOI: <http://doi.acm.org/10.1145/1342327.1342336>.
7. Hartenstein, R. The digital divide of computing. In *Proceedings of the 1st Conference on Computing Frontiers* (Ischia, Italy, Apr. 14–16, 2004). CF 2004. ACM, New York, 357–362. DOI: <http://doi.acm.org/10.1145/977091.977144>.
8. Jacobs, B. *Object Oriented Programming Oversold!*; <http://www.geocities.com/tabliizer/oopbad.htm>.
9. Wegner, P. Dimensions of object-based language design. In *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications* (Orlando, FL, Oct. 4–8, 1987). N. Meyrowitz, Ed. OOPSLA 1987. ACM, New York, 168–182. DOI: <http://doi.acm.org/10.1145/38765.38823>.

Mordechai (Moti) Ben-Ari (benari@acm.org) is an associate professor in the Department of Science Teaching at Weizmann Institute of Science in Rehovot, Israel, and an ACM Distinguished Educator.

Copyright held by author.

I do not believe there is a "most successful" way of structuring software nor that any method is "dominant."

Point/Counterpoint

Future Internet Architecture: Clean-Slate Versus Evolutionary Research

Should researchers focus on designing new network architectures or improving the current Internet?

OVER THE PAST several years, the networking research community has engaged in an ongoing conversation about how to move the field—and the Internet itself—forward. These discussions take place in the context of the tremendous success of the Internet, begging the question of whether researchers should focus on understanding and improving today's Internet or on designing new network architectures that are unconstrained by the current system. Ultimately, individual researchers have their own styles, often a unique blending of both approaches. In this Point/Counterpoint, Jennifer Rexford and Constantine Dovrolis debate the pros and cons of “clean slate” and “evolutionary” approaches to networking research, reflecting on the larger discussion taking place in the networking research community.

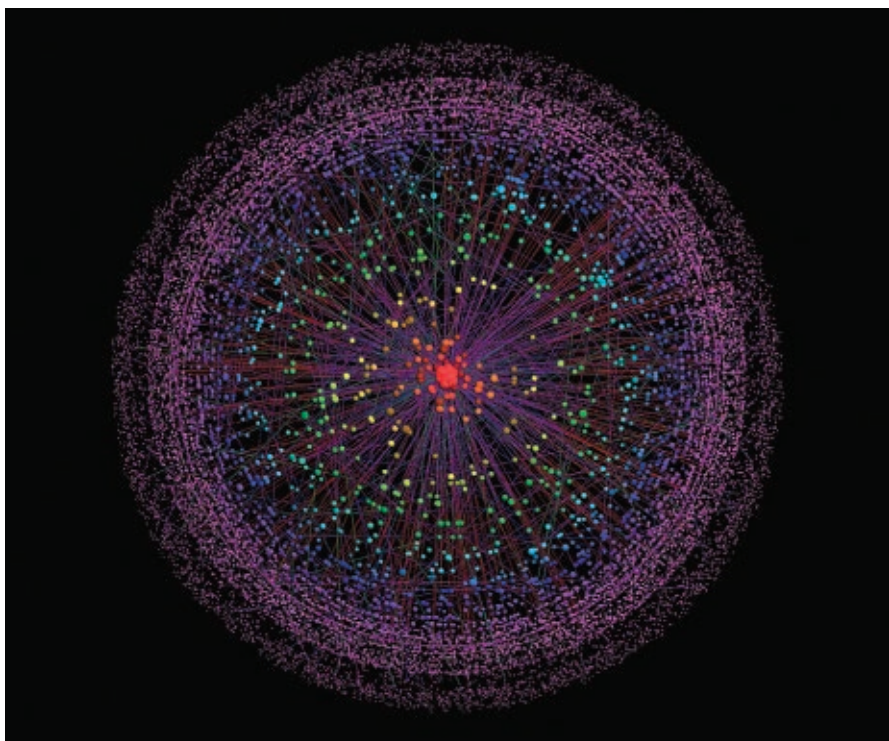
Point: Jennifer Rexford

The Internet is an undeniable success—a research experiment that escaped from the lab to become a major part of the global communications infrastructure. The seeds of the Internet's success lie in its “underspecified” design—a minimalist network providing a simple best-effort packet-delivery service coupled with programmable computers at the end points. These early design decisions were so important because they lowered the barriers to in-

novation in new applications (created by anyone who wants to program these computers) and link technologies (that can be easily adopted if they support the basic packet-delivery model). This has led to innovation far beyond what any of the early designers of the Internet could have ever imagined.

Given the Internet is so successful, and apparently so accommodating of innovation, “clean slate” networking

research may seem strange, even superfluous. Yet, nothing could be further from the truth. In fact, clean-slate design is important for enabling the networking field to mature into a true discipline, and to have a future Internet that is worthy of society's trust. Contrary to the very premise of our debate, I do not believe that evolutionary and clean-slate research are at odds. Insights from clean-slate research can



Nodal representation of the Internet.

(and should) help guide the ongoing evolution of the Internet, and a clean-slate redesign may be necessary for the Internet's continued evolution into a secure, reliable, and cost-effective infrastructure. Most importantly, as a research community, we should plant the seeds that will enable future research experiments to "escape from the lab."

Toward a Networking Discipline

The success of the Internet does not mean the field of networking is mature. Far from it. The Internet has grown and changed much faster than our own understanding of how to design, build, and operate large, federated networks. This is a common phenomenon in engineering. The great medieval cathedrals were built long before the field of civil engineering was in place. As a result, many of these early cathedrals collapsed under their own weight after decades of construction. Even the collapsed cathedrals were an invaluable learning experience along the long road toward a more rigorous approach to designing and building large structures. They were a step in the journey, not the destination itself. The way we design large buildings today reflects more than incremental improvements in engineering techniques, but a fundamentally more principled approach to the problem.

Whenever the Internet faces new challenges, from the fears of congestion collapse in the late 1980s to the pressing cybersecurity concerns of today, new patches are introduced to (at least partially) address the problems. Yet, we do not yet have anything approaching a discipline for creating, analyzing, and operating network protocols, let alone the combinations of protocols and mechanisms seen in real networks. Networking is not yet a true scholarly discipline, grounded in rigorous models and tried-and-true techniques to guide designers and operators. Witness any networking class or textbook, riddled as they are with descriptions of existing protocols rather than a top-down treatment of the "laws" or even "rules of thumb" governing the design, analysis, and operation of these protocols. Given the critical importance of communication networks, we need the field to mature into a discipline we can apply confidently in practice and teach

As a research community, we should plant the seeds that will enable future research experiments to "escape from the lab."

effectively to our students.

While studying today's Internet is clearly an important part of maturing the field, it is not enough; we also need exploration that is unfettered by today's artifacts. To be clear, ignoring today's artifacts does not mean ignoring reality. Any new designs must still grapple with practical constraints (such as the speed of light, or limitations on computation, memory, and bandwidth resources) and design requirements (for goals like efficiency, security, privacy, reliability, performance, ease of management, and so on). Yet, a clean-slate design process could remain free of the considerable minutiae of today's protocols and operational practices, and the challenges of incremental deployment.

A clean-slate design process can topple the underlying assumptions of today's architecture, such as asking whether we can achieve scalability without relying on hierarchical addressing, route traffic directly on the name of a service rather than the address of a machine, or have notions of identity that cannot be spoofed. This clean-slate exploration can lead to valuable new designs that fill out the large design space, expanding our knowledge and experience. This exploration can, perhaps more importantly, lead to new methodologies for designing networks and protocols. Whether and how to deploy these new ideas in today's Internet, while certainly a worthy topic in its own right, should sometimes be secondary to the broader goal of deepening our understanding of the field. The measure of successful

research should be the greater depth of our understanding, not just the breadth of deployment.

Yet, clean-slate networking research cannot stop at pencil-and-paper designs. In addition to new ideas, and rigorous theoretical models and analysis, we need to push our ideas further into real implementations and (ideally) deployments. The "Eureka" moments that lead to real progress happen when we encounter surprises, when something happens that we could never have planned or predicted. Building, evaluating, and deploying real systems—on experimental facilities such as the proposed GENI and Federica platforms (in the U.S. and Europe, respectively)—exposes our nascent ideas to the harsh light of day, and gives us the feedback necessary to help our ideas grow sharper and stronger as we address the unexpected setbacks and limitations, and embrace the practical constraints and design requirements we were unwittingly ignoring.

Building and deploying our designs is more than just the last step in evaluating an idea—it is part of a continuous cycle of research, constantly refining the problem, the models, and the solutions until a more complete understanding emerges. This approach to networking research should sound familiar—it is exactly how the early ARPAnet was designed and built, leading to the amazing advances we have seen in the 40 years since the first message was delivered over the network we would come to call "the Internet." At the time, the notion that the ARPAnet would eventually overtake the established telecommunication networks of its day was inconceivable to most people. But, we know now how that story turned out.

Toward an Internet Worthy of Our Trust

The Internet is showing signs of age. Pervasive security problems—spam, denial-of-service attacks, phishing, and so on—are only the most visible symptoms. The Internet also does not handle mobile hosts, whether users on the move or virtual machines migrating from one computer to another, all that well. The Internet's best-effort service model is a poor match for many real-time applications, such as IPTV and videoconferencing. The Internet is not

reliable enough, due to equipment failures, software bugs, and configuration mistakes. Managing a large network is too expensive—often costing more than the underlying equipment—and tremendously error prone. The Internet consumes too much energy, in an era of serious concern about global warming. The Internet does not seem ready to handle the coming onslaught of countless small sensor devices that have the potential to revolutionize our world. The list goes on and on.

Many of these pressing challenges are deeply rooted in early design decisions underlying the Internet, and may not be solvable without fundamental architectural change. For example, many security problems relate to the Internet's weak notions of identity, and particularly the ease of spoofing everything from IP addresses to domain names, from email addresses to routing information. Stronger notions of identity are not easily retrofitted on today's architecture. Mobility is difficult to handle because IP addresses are hierarchical and tightly coupled with the scalability of the routing protocols. Breaking this coupling may require a new relationship between naming, addressing, and routing. Network management is difficult because of the current "division of labor" between the distributed protocols running on the network elements and the management systems that can only indirectly tune the many knobs these protocols expose. Solving these problems may require us to revisit some of the most basic principles underlying the Internet of today.

Clean-slate research allows us to explore radically new designs, to see if

they are viable alternatives to the solution we have now. Some of these clean-slate solutions may very well have an incremental path to deployment. But, as the American baseball legend Yogi Berra famously said, "You've got to be very careful if you don't know where you're going, because you might not get there." Clean-slate research can help us determine where we should be going. Clean-slate design may also help us decide what parts of the Internet should *not* change. Perhaps, despite the challenges facing today's Internet, we fundamentally cannot do much better along some dimensions (say, security) without paying too high a price along some other dimension. Clean-slate research can help us understand those trade-offs, to guide decisions about whether and what to change.

Finally, perhaps wholesale change is both necessary and possible. Despite enabling innovation in applications and link technologies, the Internet architecture itself is remarkably resistant to change. In redesigning the Internet, we can direct much-needed attention to this problem. Making the inside of the network more programmable, and allowing multiple independent designs to coexist in parallel, are a promising start in this direction. Perhaps the future Internet could have the seeds for its own constant reinvention lying within it. We are already seeing the early fruits of this kind of clean-slate thinking, in software-defined networking infrastructures like OpenFlow (<http://www.openflowswitch.org/>) that are being deployed in several enterprise, datacenter, and backbone networks. Even experimental infra-

structures like GENI and Federica, designed as they are to enable multiple simultaneous experiments with new network architectures, are themselves examples of this kind of change.

Fundamental change like this is, indeed, possible and it is already starting to happen, due to the early clean-slate research efforts over the past several years. Further, more substantive change can happen in the years ahead. Given the Internet largely supplanted the circuit-switched telephone networks, is it so farfetched to think that something else might supplant the Internet, or so significantly alter the Internet that we no longer recognize it from the descriptions we see in today's networking textbooks?

Conclusion

Networking is still a young field. While the Internet's success is something we should admire and celebrate, we should not be content with our current understanding of the field or view the Internet architecture as set in stone. Perhaps a new generation of researchers and practitioners will turn the future Internet into something that only vaguely resembles its predecessor. Perhaps this future network will accommodate change more broadly and deeply than even today's Internet has. A willingness to step back, and design from scratch, is an important part of the research repertoire that can enable these advances in the field, and of the Internet itself. 

Jennifer Rexford (jrex@cs.princeton.edu) is a professor in the computer science department at Princeton University in New Jersey.

Copyright held by author.

Counterpoint: Constantine Dovrolis

LET US FIRST identify the major difference between the two approaches. Evolutionary Internet research aims to understand the behavior of the current Internet, identify existing or emerging problems, and resolve them under two major constraints: first, backward compatibility (interop-

erate smoothly with the legacy Internet architecture), and second, incremental deployment (a new protocol or technology should be beneficial to its early adopters even if it is not globally deployed).

On the other hand, clean-slate research aims to design a new "Future Internet" architecture that is significantly better (in terms of performance, security, resilience, and other properties) than the current Internet without

being constrained by the current Internet architecture.

Clean-Slate Research and Its Real-World Impact

Clean-slate Internet research is not something new. In fact, there is a long history of such efforts and we can learn something by analyzing whether earlier clean-slate protocols and architectures have been adopted or not. To name a few examples, consider active

networks, per-flow QoS guarantees and admission control, the connectionless network protocol CLNP, transport protocols such as XCP, or interdomain routing architectures such as Nimrod. There is also a large number of protocols that are more or less backward compatible but not truly incrementally deployable, such as IPv6, interdomain IP multicast, RSVP, and IntServ, IPsec, or S-BGP. Arguably, these protocols have not seen large-scale deployment, at least so far. The “real world” adopted instead evolutionary approaches such as NATs, caching and content distribution networks, DiffServ, adaptive applications, and various security mechanisms (such as end-host security, intrusion detection systems, and routing filters) that work well with the legacy architecture. Why does clean-slate architectural research, or even protocols and designs that attempt to be backward compatible, often fail to be adopted in practice?^a

In industrial economics, it is well known that an emerging technology that is subject to network externalities will probably not be able to replace a widely deployed but inferior technology, as long as there are costs involved in switching from the incumbent to the emerging technology (see Arthur¹ and related papers). Instead, the more relevant question is whether the emerging technology offers a valuable new service the current technology cannot

a I do not claim that the research on those earlier clean-slate protocols was mediocre or that it did not have academic impact—I am strictly focusing on their deployment and real-world impact.

How does the additional value of a new technology, relative to the incumbent technology, compare to the transition cost?

The ARPANET architecture was only one of several competing architectures and it was through a long evolutionary process that it prevailed.

provide directly or indirectly. In other words, how does the additional value of a new technology, relative to the incumbent technology, compare to the transition cost?

It is not enough for a clean-slate architecture to be “better” than the current Internet architecture. For the former to have real impact it should be able to replace the latter—otherwise it will remain an intellectual exercise. It is the question of real-world impact that differentiates clean-slate from evolutionary research and design. And at least so far, the proponents of clean-slate research have not shown instances of such new applications or services that cannot be directly or indirectly constructed for the current Internet. Incidentally, the promise of a “secure and trustworthy Future Internet” is appealing but not convincing: there is no way to provide security guarantees with an open-ended threat model. Further, it is very likely that a brand-new internetworking architecture will have more design and implementation bugs and security holes than the current Internet architecture (which is being “debugged” for more than 30 years now).

The proponents of clean-slate design emphasize they will not stay with “paper designs”—they will build and experiment with the proposed architectures in testbeds such as GENI. But what would that prove? Several previous clean-slate protocols were also implemented and tested 10 or 20 years ago. The issue was not the lack of implementation or experimentation, but the fact that those protocols could not compete with incumbent

technologies, considering the actual benefits they provide to users and the costs involved in the technological transition. These are issues of mostly economic nature that GENI or other testbeds cannot help us study. Further, these testbeds are not used by real applications and people and they do not operate under the economic and policy constraints of the real world. The early ARPANET succeeded because it was not just a testbed: it was also used as a production network, connecting some universities and research labs, while at the same time networking researchers could experiment with new protocols and technologies.

Another popular claim is that the current Internet architecture is the result of clean-slate thinking back in the 1960s or 1970s. However, we should not ignore that packet switching or TCP/IP were not inventions that “came out of nowhere”—they resulted from an evolutionary process that started from synchronous multiplexing in circuit-switched networks, moving to asynchronous multiplexing and then to datagram forwarding. Further, the ARPANET architecture was only one of several competing architectures (such as IBM SNA, DECnet, ITU X.25, Xerox Pup, SITA HLN, or CYCLADES), and it was through a long evolutionary process that the former eventually prevailed.

Is the Internet Architecture Really “Ossified”?

One of the primary arguments for clean-slate research has been that the current Internet architecture is ossified, especially at the central layers of the protocol stack (IP and TCP), and that ISPs have no incentive to adopt any architectural innovations. This is a rather negative view of what happens. The Internet architecture maps an ever-increasing diversity of link-layer technologies to a rapidly increasing range of applications and services. To support this innovation at the lowest and highest layers of the architecture, the central protocols of the architecture must evolve very slowly so that they form a stable background on which diversity and complexity can emerge.

To use a biological analogy, certain developmental Gene Regulatory Networks were established in the Early

Cambrian (about 510 million years ago) and they have not evolved significantly since then. These GRNs are referred to as evolutionary kernels, and it is now understood that they are largely responsible for major aspects of all animal body plans. For instance, the heart of a fruit fly and the heart of a human, despite distinct morphologies, develop using the same core cardiac GRN. Evolutionary kernels represent a stable basis on which diversity and complexity of higher-level processes can evolve.²

An Agenda for Evolutionary Internet Research

Instead of thinking about the Internet as an artifact that we designed in the past and we can now redesign, we can start thinking of the Internet as an evolving ecosystem that is affected by, and in turn is affecting, several disciplines and how we study them. Its evolution is controlled, not only by technology, but also by the global economy, creative ideas by millions of individuals, and a constantly changing set of “environmental pressures” and constraints. Our mission then, as Internet researchers, is to first measure and understand the current state of this ecosystem, predict where it is heading and the problems it will soon face, and create what could be referred to as intelligent mutations: innovations that can, first, avoid or resolve those challenges, and second, innovations that can be adopted by the current architecture in a way that is backward compatible and incrementally deployable. This is a pragmatic research agenda that can have real impact on millions of people.

Instead of testbeds, evolutionary research needs various experimental resources that will be integrated in the current Internet. First, we need a dense infrastructure of “Internet monitors” of various types that will allow us to accurately measure what is currently happening in this evolving ecosystem. It is embarrassing that (despite the tremendous value of the Route Views project) we still do not have an accurate way to measure the Internet interdomain topology. We also do not have an estimate of how much traffic flows between any two autonomous systems, even though that interdomain traffic matrix largely determines the economics of the global Internet. Plus, we have no way to know

We can start thinking of the Internet as an evolving ecosystem that is affected by, and in turn is affecting, several disciplines and how we study them.

how the Internet population uses the Internet and the Web across time and space. As this knowledge gap increases, I am concerned we will soon be unable to track our own creation, and much more to influence its future.

Together with an extensive monitoring infrastructure, evolutionary Internet research would greatly benefit if we could operate our own experimental ISP. This would be a real TCP/IP network, running all protocols of the current Internet architecture, present at many Internet Exchange Points, peering openly with other ISPs and content providers, and carrying traffic that belongs to real Internet users. One way to do so could be that universities use this experimental ISP to carry part of their traffic for free, with the understanding that this is a research network and so its traffic may be subject to experimental “mutations” of the Internet architecture. This is different than Internet2 or NLR, which are production networks, and certainly very different than isolated GENI-like testbeds.

Where is the Science, After All?

The proponents of clean-slate design claim their approach leads to a science of network design (sometimes referred to as “network science,” which is confusing because the same term is used in other disciplines to refer to the study of complex systems using dynamic graph models and network analysis techniques). It is also often claimed that evolutionary Internet research is not a science, but a collection of “hacks” and incremental improvements. This is a misleading position.

Several breakthroughs in networking research resulted from evolutionary research. For instance, major results in congestion control and active queue management resulted from attempts to understand and improve TCP, the discovery of fundamental properties of the Internet traffic and topology, the design of innovative peer-to-peer communication protocols, or the development of end-to-end network inference as well as network tomography methods.

A domain of knowledge does not become science because it is based on clean optimization frameworks or because it proves deep results about toy models. Good science requires relevance to the real world, measurements and experimental validation, testable hypotheses, and models with predictive power.

Epilogue

I often wonder, what is the main reason that well-respected Internet researchers have decided to pursue the clean-slate approach? It cannot be just the “funding carrot,” I am sure. Here is one possible answer from a science fiction TV series. In “Battlestar Galactica” (S4-E21), Mr. Lampkin says to Commander Adama: “I have to say I’m shocked with how amenable everyone is to this notion of (...leaving everything behind and starting with nothing on the newly discovered planet Earth).” Commander Adama responds “Don’t underestimate the desire for a clean slate, Mr. Lampkin.” It may be that we find joy and pride in the idea that we can redesign the Internet from scratch, that we can avoid all previous mistakes and do it perfectly this time. If we do not want to sound like science fiction dialogue, however, it is important that we continue to foster the evolution of the current Internet, having positive impact on the way many millions of people live, work, and communicate. ■

References

1. Arthur, W.B. Competing technologies, increasing returns, and lock-in by historical events. *The Economic Journal* 99, 394 (1989), 116–131.
2. Dovrolis, C. and Streedman, T. Evolvable network architectures: What can we learn from biology? *ACM SIGCOMM Computer Communications Review (CCR)* 40, 2 (Apr. 2010).

Constantine Dovrolis (dovrolis@cc.gatech.edu) is an associate professor in the College of Computing at Georgia Tech in Atlanta, GA.

Copyright held by author.

Call for Nominations

The ACM Doctoral Dissertation Competition

Rules of the Competition

ACM established the Doctoral Dissertation Award program to recognize and encourage superior research and writing by doctoral candidates in computer science and engineering. These awards are presented annually at the ACM Awards Banquet.

Submissions

Nominations are limited to one per university or college, from any country, unless more than 10 Ph.D.'s are granted in one year, in which case two may be nominated.

Deadline

Submissions must be received at ACM headquarters by **October 31, 2010** to qualify for consideration.

Eligibility

Each nominated dissertation must have been accepted by the department between October 2009 and September 2010. Only English language versions will be accepted. Please send a copy of the thesis in PDF format to emily.eng@acm.org.

Sponsorship

Each nomination shall be forwarded by the thesis advisor and must include the endorsement of the department head. A one-page summary of the significance of the dissertation written by the advisor must accompany the transmittal.

Publication Rights

Each nomination must be accompanied by an assignment to ACM by the author of exclusive publication rights. (Copyright reverts to author if not selected for publication.)

Publication

Winning dissertations will be published by Springer.

Selection Procedure

Dissertations will be reviewed for technical depth and significance of the research contribution, potential impact on theory and practice, and quality of presentation. A committee of five individuals serving staggered five-year terms performs an initial screening to generate a short list, followed by an in-depth evaluation to determine the winning dissertation.

The selection committee will select the winning dissertation in early 2011.

Award

The Doctoral Dissertation Award is accompanied by a prize of \$20,000 and the Honorable Mention Award is accompanied by a prize of \$10,000. Financial sponsorship of the award is provided by Google.

For Submission Procedure

See <http://awards.acm.org/html/dda.cfm>



Association for
Computing Machinery



Article development led by [acmqueue](http://acmqueue.queue.acm.org)
queue.acm.org

Information technology has the potential to radically transform health care. Why has progress been so slow?

BY STEPHEN V. CANTRILL, M.D.

Computers in Patient Care: The Promise and the Challenge

A 29-YEAR-OLD FEMALE from New York City comes in at 3 A.M. to an emergency department (ED) in California, complaining of severe acute abdominal pain that woke her up. She reports that she is visiting California to attend a wedding and that she has suffered from similar abdominal pain, most recently resulting in an appendectomy. The emergency physician performs an abdominal CAT scan and sees what he believes to be an artifact from the appendectomy in her abdominal cavity. He has no information about the patient's past history other than what she is able to tell him; he has no access to any images taken before or

after the appendectomy, nor does he have any other vital information about the surgical operative note or follow-up. The physician is left with nothing more than what he can see in front of him. The woman is held overnight for observation and released the following morning symptomatically improved, but essentially undiagnosed.

A vital opportunity has been lost, and it will take several months and several more physicians and diagnostic studies (and quite a bit more abdominal pain) before an exploratory laparotomy will reveal that the woman suffered from a rare (but highly curable) condition, a Meckel's diverticulum. This might well have been discovered that night in California had the physician had access to complete historical information.

This case is recent, but the information problem at its root seems a holdover from an earlier age: Why is it that in terms of automating medical information, we are still attempting to implement concepts that are decades old? With all of the computerization of so many aspects of our daily lives, medical informatics has had limited impact on day-to-day patient care. We have witnessed slow progress in using technology to gather, process, and disseminate patient information, to guide medical practitioners in their provision of care and to couple them to appropriate medical information for their patients' care.

Why has progress been so slow? Some of the delay certainly has been technologically related, but not as much as one might think. This article looks at some of these issues and the challenges (there are many) that remain.

First, why bother with computers in health care, anyway? There are many potential advantages from the application of health information technology (or HIT, the current buzzword). These include improved communication between a single patient's multiple health-care providers, elimination of needless medical testing, a decrease in medical errors, improved qual-



ity of care, improved patient safety, decreased paperwork, and improved legibility (yes, it's still an issue). Many of these improvements have not yet come to pass and many others are nearly impossible to rigorously prove, but for the purposes of this discussion, let's assume that HIT is a good thing.

Some History

The first challenge in applying medical informatics to the daily practice of care is to decide how computerization can help patient care and to determine the necessary steps to achieve that goal. This challenge is best summed up by

Lawrence L. Weed, M.D.: to develop an information utility that has currency of information and parameters of guidance to assist medical personnel in caring for and documenting the care of patients.³ From the technology side, we need a facile interface between human and machine and a responsive, reliable system that is always available. The assumption is there will be adequate computational power and mass memory to support such a system.

The history of the computer industry's involvement in these problems is instructive. In the late 1960s, a major computer vendor thought it could

solve many hospital-based medical care issues in less than a year by deploying 96-button punch pads throughout the hospital to handle physician orders and intra-hospital communication. Button-template overlays were to be used to support different types of orders. As it turned out, this was a most inadequate human interface: cumbersome, inflexible, close-ended with limited duplex communication, and so on. Not surprisingly (at least to the users), this was a nonstarter and failed.

Most of the major hardware vendors of that era also had plans to provide

automation of hospital information, creating their versions of a hospital information system (HIS). For various reasons, they all failed, often with a stunning thud. The most commonly cited deficiencies were a poor human interface, unreliable implementation, and cost. As is often the case when applying new technology to a discipline, the magnitude and complexity of the problem was initially grossly underestimated. As a result, most hardware vendors then limited themselves to the historic area for data processing: patient billing and the financial arena.

In the late 1960s and early 1970s, hardware limitations strained even demonstration systems. Limited main and mass memory, CPU speed, and communication between the CPU and user workstations were all factors that limited system usability and capacity. The human-machine interface was also an issue. Some systems used light-pens with some degree of success.

At that time, I was a member of a small group that was implementing a demonstration of an electronic medical record system that used touch-sensitive screens: a 24-line by 80-character CRT display that allowed two columns of 12 text selections each to be presented to the user, with a branch taken to a new display based upon the selection. This “branched-logic” approach allowed medical users to concatenate a series of selections to create complex text entries for storage into a patient’s medical record, as well as to order medications and lab tests and retrieve previous entries from the patient’s medical record.^{1,2} (The ability to type in information was supported for those situations where the displays did not contain the desired medical content.) The major performance goal of this system (and its 20 workstations) was to provide a new text display to any user within 300 milliseconds at least 80% of the time, which was quite advanced for its time. This system was designed to be available around the clock with no scheduled downtime.

This demonstration system presented several challenges. First and foremost was the interface between machine and medical providers (physicians, nurses, and so on), as well as patients (for entering their own medical histories). Medicine as a discipline

is not known to be necessarily forward-looking in adopting new technology, so convincing these individuals to use a revolutionary technology to replace pen and paper was not easy.

The mass-storage limitations were real. The system would support only 144 active patients at any one time (which was adequate for operation on a single hospital ward but would preclude initially supporting an entire hospital). There was also a limit of 32K individual text screens of information (fancy that!), and there were limits on how far the dumb terminals could be placed from the CPU.

This demonstration system was able to support an entirely computerized medical record (now called an electronic medical record, or EMR) and allowed physicians to use the touch-screen and branched logic displays to enter a patient’s history, physical examination, problem list (those unique medical issues for each patient), and progress notes, including patient assessments and orders. For many specific problems, the system would offer a range of recommended treatments (for example, the appropriate list of drugs for hypertension). As part of the physician-ordering sequence for each specific drug, the system would present side effects to watch for, recommended drug monitoring parameters, drug-drug interactions, among other features. (This is an obvious precursor to having this checking done automatically). This level of guidance was possible because of the structured nature of the data entry; it is much more difficult when free text is entered via the keyboard instead.

Why didn’t this demo system catch on as hardware and operating systems improved? There were several reasons. At the time, computers were not well understood and, thus, were considered a bit intimidating by the general public, so there was a degree of user hesitation. Also, the level of medical documentation needed and the support of patient safety issues this system was based upon were not, unfortunately, appreciated at that time. Cost also continued to be an issue. Although this system never caught on, many of the concepts it demonstrated are present in currently evolving commercial systems.

Several other early attempts were

made to apply computerization to health care. Most were mainframe-based, driving “dumb” terminals. Many dealt only with the low-hanging fruit of patient order entry and results reporting, with little or no additional clinical data entry. Also, many systems did not attempt to interface with the information originator (for example, physician) but rather delegated the system use to a hospital ward clerk or nurse, thereby negating the possibility of providing medical guidance to the physician, such as a warning about the dangers of using a specific drug. This is a nontrivial issue that still is a problem with some systems today, illustrating the challenge of an effective user interface.

There were also some efforts to automate the status quo with no attempt to structure the data input. This usually meant having the health-care provider enter free text via a keyboard. Unfortunately, this automation of unstructured data yields only (legible) unstructured data. This may be acceptable when dealing with a system of limited scope but does not work well with massive amounts of information such as a patient record.

These computer systems were quite expensive to install and operate. With this foray into the clinical realm of acute medical care, the requirements for increased reliability of both hardware and software became clear, along with the need for constant accessibility.

Areas of Real Technical Progress Over the Years

We have made significant technological advances that solve many of these early shortcomings. Availability of mass storage is no longer a significant issue. Starting with a 7MB-per-freezer-size-disk drive (which was not very reliable), we now have enterprise storage systems providing extremely large amounts of storage for less than \$1 per gigabyte (and they don’t take up an entire room). This advance in storage has been accompanied by a concomitant series of advances in file structures, database design, and database maintenance utilities, greatly simplifying and accelerating data access and maintenance.

The human-machine interface has seen some improvement with the evolution of the mouse as a pointing de-

vice and now the partial reemergence of the touchscreen. We have also seen the development of the graphical user interface, which has facilitated user multitasking.

Overall system architectures have followed an interesting course: from a centralized single CPU and dumb workstations to networks with significant processing capabilities at each workstation. In some situations we are now seeing a movement to the so-called thin-client architecture, again with limited power and resources at each workstation, but a significant improvement in ease of system maintenance.

Of course, all of this has been made possible by improvements in transmission speed of data both between systems and within a single network. These advances in potential system responsiveness, however, have been attenuated by the ever-increasing computational demands of the software, sometimes legitimately, but often caused by the proliferation of bloatware: cumbersome, poorly designed, and inefficiently coded software serving as a CPU-cycle black hole.

An additional complicating factor has been the migration of many pieces of application software to Web-based processes. This does provide the advantage of platform semi-independence, but any slowness of the browser or the Web server is inflicted on the user, and in some cases, may be a dealbreaker in terms of user acceptance. For example, say I use a Web-based system to order a series of medications on a patient and it takes me 10 mouse clicks/screen flips to order a single medication. If it takes one second to move from screen to screen, that is 10 seconds (plus my human processing time). Not bad for a single order, but multiply that by 20 orders per patient over 10 sick patients in a busy emergency department at 1 A.M. on a hectic Saturday night, and you begin to appreciate the issue. There are approaches to minimize this negative impact, but these require a degree of sophistication of system design that is not always present. In fact, a common complaint of medical users is that "it's too many clicks to do something simple."

A very significant area of technological improvement has been in the acquisition, processing, transmission,



With all of the computerization of so many aspects of our daily lives, medical informatics has had limited impact on day-to-day patient care.



and presentation (display) of graphical images. This capability has, over the past decade, given us increasingly sophisticated CAT scan and MRI results and has allowed most hospitals to discontinue the use of X-ray film almost completely, using digitally stored images instead. These picture archiving storage (PACS) systems have revolutionized radiology and improved patient care by allowing easy distribution of these images to all care providers of a specific patient, alleviating the endless problem of trying to chase down the original physical X-ray film.

Areas of Limited Progress

If we truly want to develop an information utility for health-care delivery in an acute care setting (such as an intensive care unit or emergency department), we must strive for overall system reliability at least on the order of our electric power grid, ideally with no perceived scheduled or unscheduled downtime. Some health-care information computer systems have achieved a high degree of reliability, but many have not. These lower-performing systems often had their beginnings, as noted earlier, in non-mission-critical applications such as patient billing. This, unfortunately, established a system culture that is permissive of system failure, and this culture is difficult to upgrade.

The culture of system reliability begins with the hardware architecture and progresses through the operating system, the application programs, and the supporting institutionwide infrastructure, physical deployment, and extensive failure mode analysis. This means simple things such as supporting rolling data backups and system updates without taking the system down (from the user's point of view). Some systems boast they have uptimes of 99.99%, but that means they are still unavailable for an hour per year.

Reliability and availability remain ongoing challenges. Certainly, manual procedures for use during system unavailability are necessary, but the goal should be not to have to use them. This is an increasingly important issue as we attempt to develop systems that are more intimately involved in patient care (such as online patient monitoring of vital signs and real-time patient

tracking). In fact, we should not even attempt to support mission-critical operations unless we have the hardware, software, and support systems in place that will guarantee extreme overall reliability. Even then it is a risk. I remember the promises from our “state of the art” enterprise RAID 5 storage vendor: “It will never go down.” These promises were used to convince me to move off my dual-write standby server configuration to the enterprise storage system to serve up block storage for my emergency department network. This system is critical, providing real-time ED patient tracking, clinical laboratory result access, patient-care protocol information, emergency department log access, hospital EMR retrieval, metropolitan area hospital ambulance divert status, and physician and nurse order communication, among other functions. Unfortunately, the storage system that was promised to “never go down” had two five-hour failures over a two-year period, thoroughly dispelling the myth of reliability promised by the vendor. These episodes, unfortunately, are not unique. Through careful design and adequate component redundancy, we have been able to achieve high levels of reliability in safety-critical systems; our patients and health-care providers deserve no less reliability.

Patient data entry in any health information system is labor intensive. Health-care providers (especially physicians) have little tolerance for systems that serve as impediments to getting their work done, often regardless of what positives might accrue from using such a system. This represents a failure of interface and software design and may explain why we are seeing increased use of “scribes” in institutions that have implemented electronic health records. These scribes are individuals who act as recorders for the health-care professionals so they do not have to interface directly with the computer system. Obviously, this greatly diminishes the power of any system since there is no longer an interface with the information originator. The incorporation of dynamic medical guidance (advice rules based upon individual patient data such as checking a drug order for interactions with the patient’s other drugs) is of limited utility if the data is



Advancements in storage have been accompanied by a concomitant series of advances in file structures, database design, and database maintenance utilities, greatly simplifying and accelerating data access and maintenance.



entered by someone other than the information originator.

It is also interesting to note that many institutions that had early success with even poorly designed systems were those where the majority of the care was supplied by physicians in training. They were told to use the system “or else” and did not have the flexibility to move to another institution. To maximize user acceptance of any system, we need to continue to improve the human-machine interface, allowing for branched logic content, templated data entry, voice recognition, dynamic pick lists, and when absolutely necessary, free text entry. Physicians care greatly about their patients; if an institution’s attempts at computerization do not result in improved patient care and/or improved speed or other significant advantages, acceptance of any system will be problematic. This issue has resulted in the demise of many hospital-based systems.

Even where successfully implemented, computerized health information systems have sometimes had unanticipated side effects. One significant issue is the explosion of data that may be stored in the patient record. This can quickly escalate beyond the capability of the human mind. The challenge remains how best to present the data to a health-care provider in an efficient and comprehensive fashion.

Another potential problem with electronic medical records is abuse of privacy. With old paper medical records, control was somewhat easier: unless copied, they were in only one place at one time. This barrier is removed with computerization, mandating enhanced restrictions to protect data. Unfortunately, we have witnessed several instances of inappropriate access to an individual’s medical data. This is most commonly seen when a celebrity is hospitalized and human curiosity results in patient privacy violations (and often subsequent firings). The challenge is to limit inappropriate access but not make legitimate data retrieval burdensome or difficult.

Ongoing Barriers in the Success of HIT

As we continue to strive for advances in health information technology, we must confront several barriers to

its success. One significant issue is the “balkanization” of medical computerization. Historically, there has been little appreciation of the need for an overall system. Instead we have a proliferation of systems that do not integrate well with each other. For example, a patient who is cared for in my emergency department may have his/her data spread across nine different systems during a single visit, with varying degrees of integration and communication among these systems: emergency department information system (EDIS), prehospital care (ambulance) documentation system, the hospital ADT (admission/discharge/transfer) system, computerized clinical laboratory system, electronic data management (medical records) imaging system, hospital pharmacy system, vital-signs monitoring system, hospital radiology ordering system, and PACS system. Ideally, these different systems should be integrated into a seamless whole, at least from the user’s point of view, but each has a different user interface with different rules, a different feel, and different expectations of the user. It really is just a bunch of unconnected pieces, which may, in certain situations, actually increase the time and effort for patient care. In this case, the full capability of data integration clearly has not been achieved.

This leads to other concerns: Are we creating health-care computer systems that are so complex that no one has a complete understanding of their vulnerabilities, thus making them prone to failure? Do we have an adequate culture of mission-critical and fault-tolerant design and system support to achieve expected levels of reliability in all hospitals that attempt a high degree of computerization? Is there sophisticated failure analysis to ensure growth, improvement, and success in all of these institutions? Or will the tolerance for unexplained failure actually pose a risk to our patients?

As mentioned, most of these component systems have a medical content piece, as well as a technology piece. It is this creation of the medical logic and structured content in many of these systems (especially the EMR systems) that remains a time-consuming and exacting process, often requiring many

person-years of effort for a single institution. Unfortunately, because of the perceived differences in practice patterns among different locales, institutions, and physician groups, only a modicum of the work done in any one location is applicable to other locations. There should be efforts to standardize some of these differences to allow more synergy between locations and products.

Although grand claims are often made about the potential improvements in the quality of care, decreases in cost, and so on, these are very difficult to demonstrate in a rigorous, scientific fashion. Fortunately, the body of positive evidence is slowly increasing, although there are occasional signs of adverse effects resulting from computerized patient data systems. For example, there is evidence that it may be easier to enter the wrong order on the wrong patient in a computerized system than in an old hard-copy, manual system.

The Future

Although difficult to scientifically prove, the benefits from an EMR and the attendant methodologies to create and maintain it are potentially significant. Yet, we have not come very far conceptually in the past several decades in realizing the potential. Nonetheless, I feel the future is quite bright for several reasons.

First and foremost, the federal government has championed these concepts with promises of fiscal support for individual physicians and institutions that implement the concepts in a meaningful way within a specific timeframe. Second, the use of computers in most aspects of our daily lives has become commonplace, resulting in increased computer literacy and decreasing hostility to their use in a medical environment. Third, with increased national emphasis on patient safety and quality of medical-care indicators, computerization of health care offers the best and easiest approach to provide the parameters of medical guidance and allow appropriate data capture to comply with these initiatives (which will be ongoing and increasing in number and complexity).

The achievement of desired goals, however, will continue to provide a

challenge to system creators and implementers. They have the difficult job of designing, developing, and supporting systems that provide improved reliability and responsiveness and a facile human-machine interface with the knowledge and guidance to provide better health care to our citizens.

Let us return to the 29-year-old patient with acute abdominal pain in the California emergency department, now under an improved computerized health-care system. The physician in California has instant access to the operative note and medical workup for the appendectomy done many months before. This reveals that, in fact, no radiographs were taken prior to the surgery, which was done laparoscopically. This implies the finding on the CAT scan is not, because of the surgical technique, an artifact, but an abnormal finding. This would lead in short order to surgical consultation and surgical repair, markedly decreasing the patient’s period of morbidity and suffering. Such improvements are the promise of integrating computers in patient care. With effort and skill, I feel we can meet this challenge. **C**

Related articles on queue.acm.org

Better Health Care Through Technology

Mache Creeger

<http://queue.acm.org/detail.cfm?id=1180186>

A Requirements Primer

George W. Beeler and Dana Gardner

<http://queue.acm.org/detail.cfm?id=1160447>

References

1. Schultz, J.R. 1988. A history of the PROMIS technology: An effective human interface. *A History of Personal Workstations*, A. Goldberg, ed. ACM Press. Addison-Wesley Publishing Co., Reading, MA.
2. Schultz, J.R., Cantrill, S.V., Morgan, K.G. 1971. An initial operational problem-oriented medical record system—for storage, manipulation and retrieval of medical data. In *Proceedings of AFIPS* 38.
3. Weed, L.L., M.D. 1972. Problem-oriented system. *Background Paper for Concept of National Library Displays*. J.W. Hurst and H.K. Walker, ed. Medcom Press.

Stephen V. Cantrill, M.D. has been practicing emergency medicine for more than 30 years. He recently retired from the position of associate director of emergency medicine at Denver Health Medical Center, a safety-net Level I trauma center, where he developed and supported the Emergency Medical Services Information System (EMeSIS). He is an associate professor of emergency medicine at the University of Colorado. He first starting writing code related to medical informatics in 1966 at Brown University, where he received his A.B. in physics, and he helped develop one of the early computerized medical record systems while working with Dr. Lawrence L. Weed at PROMIS Laboratory.

© 2010 ACM 0001-0782/10/0900 \$10.00

**Error-detection and correction features
are only as good as our ability to test them.**

BY STEVE CHESSIN

Injecting Errors for Fun and Profit

“That which isn’t tested is broken.” —Author unknown
“Well, everything breaks, don’t it, Colonel.”
—Monty Python’s Flying Circus

IT IS AN unfortunate fact of life that anything with moving parts eventually wears out and malfunctions, and electronic circuitry is no exception. In this case, of course, the moving parts are electrons. In addition to the wear-out mechanisms of electromigration (the moving electrons gradually push the metal atoms out of position, causing wires to thin, thus increasing their resistance and eventually producing open circuits) and dendritic growth (the voltage difference between adjacent wires causes the displaced metal atoms to migrate toward each other, just as magnets will attract each other, eventually causing shorts), electronic circuits are also vulnerable to background radiation. These fast-moving charged particles knock electrons out of their orbits, leaving ionized trails in their wake.

Until those electrons find their way back home, a conductive path exists where there once was none.

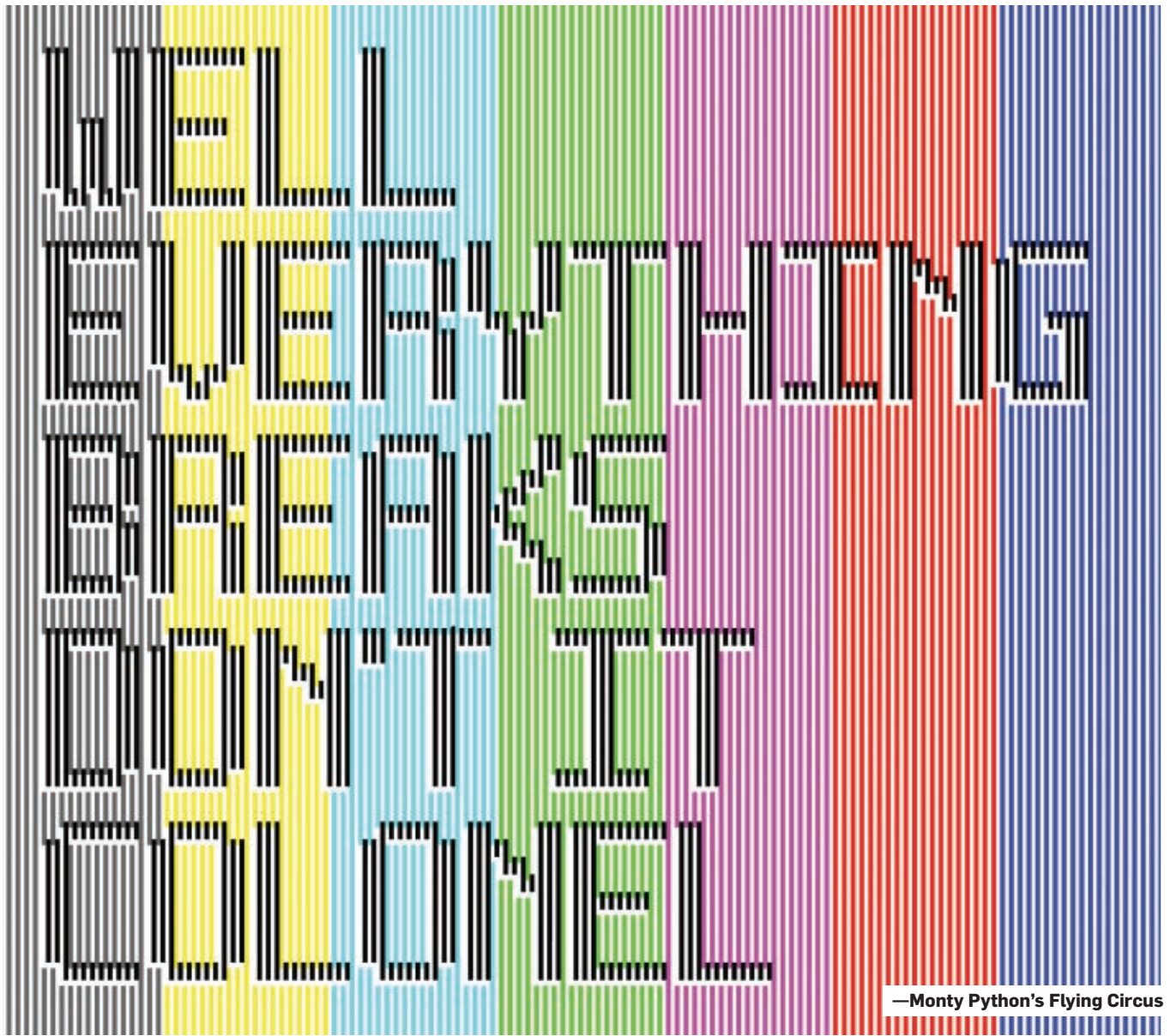
If the path is between the two plates of a capacitor used to store a bit, the capacitor discharges, and the bit can flip from one to zero or from zero to one. Once the capacitor discharges, the displaced electrons return home, and the part appears to have healed itself with no permanent damage, except perhaps to the customer’s data. For this reason, memory is usually protected with some level of redundancy, so flipped bits can be detected and perhaps corrected. Of course, the error-detection and correction circuitry itself must be tested, and that is the main topic of this article.

(If the path is between a current source and ground, then it cannot heal until power is removed. This is called *single event latchup*, which simulates a hard failure, at least until the power is turned off, such as when preparing to remove and replace the apparently failing part. The returned part, of course, will test out as “no trouble found,” frustrating everyone involved. Single event latchup is difficult for software to deal with and will not be discussed further here.)

In addition to the causes of errors mentioned here, transmission lines are subject to noise-induced errors, so transmitted signals are also often protected with redundancy.

As the density of circuits increases, features get smaller; as frequencies increase, voltages get lower. These trends combine to reduce the amount of charge used to represent a bit, increasing the sensitivity of memory to background radiation. For example, the original UltraSPARC-I processor ran at 143MHz and had a 256KB e-cache (external cache). The cache design used simple byte parity to protect the data, which was sufficient as the amount of charge used to hold a bit was large enough that an ionizing particle would drain off only a small amount, not enough to flip a bit.

When this design was scaled up in the UltraSPARC-II processor to run at 400MHz with an 8MB e-cache, however, the amount of charge used to hold a bit



—Monty Python's Flying Circus

was so small that background radiation would easily flip bits, producing on average one flipped bit per processor per year. While that might not seem like a high rate, a customer with 12 systems of 32 processors each would on average experience one failure a day. This is what led to Sun's infamous e-cache parity crisis of 1999 (more on this later; for fun, do a Web search on "e-cache parity").

Since errors, whether transient or permanent, are a fact of life, the system designers in Oracle's Systems organization (what used to be portions of Sun Microsystems) have developed a layered approach to deal with them. At the lowest level is the hardware error-detection circuitry, which records information about the error so that upper-layer software can determine if the error is transient or permanent, or

if the rate of transient errors indicates a failing part. The next layer is error correction, which can be performed by hardware, software, or a combination of the two. The third layer is diagnosis, where the Predictive Self-Healing function of the Solaris operating system determines whether a faulty part is causing the error, and whether that part should be replaced. The final level is error containment, invoked by Predictive Self-Healing when a hard failure can be fenced off so that the system can continue to function with minimal performance degradation, avoiding a disruptive and thus expensive service call.

One always hopes that errors are rare. When they do occur, however, one wants the various layers of detection, correction, diagnosis, and containment to perform flawlessly. Ensuring that requires

testing the various layers, preferably in an end-to-end fashion that imitates the behavior of real errors. Because (as one hopes) errors are rare (if they aren't, you have other problems), waiting around for them to occur naturally is not an efficient testing methodology. Thus, the need for an error injector.

An error injector requires hardware support, because during normal operation hardware only writes good data. (Without hardware support, you can simulate errors by feeding error reports to the upper layers of software, but then you aren't testing the hardware error detectors.) Hardware designers understand this, so they usually provide some means for injecting errors so that they can test their detectors. They don't always understand the environments in which errors will be injected, however.

For example, from the perspective of the hardware designer, testing the detectors during the very controlled environment of power-on self-test (POST) is sufficient, so it isn't a big deal if injecting an error has a side effect of corrupting unrelated data or destroying cache coherency. For the software designer, however, such side effects can render the error-injection hardware useless, or severely restrict the kinds of errors he or she can safely inject.

For example, while the hardware error detector does not care if a cache parity error is detected on a clean or dirty cache line, or by a user instruction or a kernel instruction, the software

by a load instruction or an instruction fetch, so recovery even from an error on a clean cache line was not possible. We were able to recover from parity errors detected by some write-backs, and we definitely improved the kernel's messages when parity errors were encountered. We prototyped confining errors that affected only a user program and not the kernel to just that program (a feature that had to wait for the System Management Facility of Solaris 10 and its process restarter before we could deploy it safely), and we introduced a cache scrubber that used diagnostic accesses to proactively look for parity errors on clean cache lines in a safe

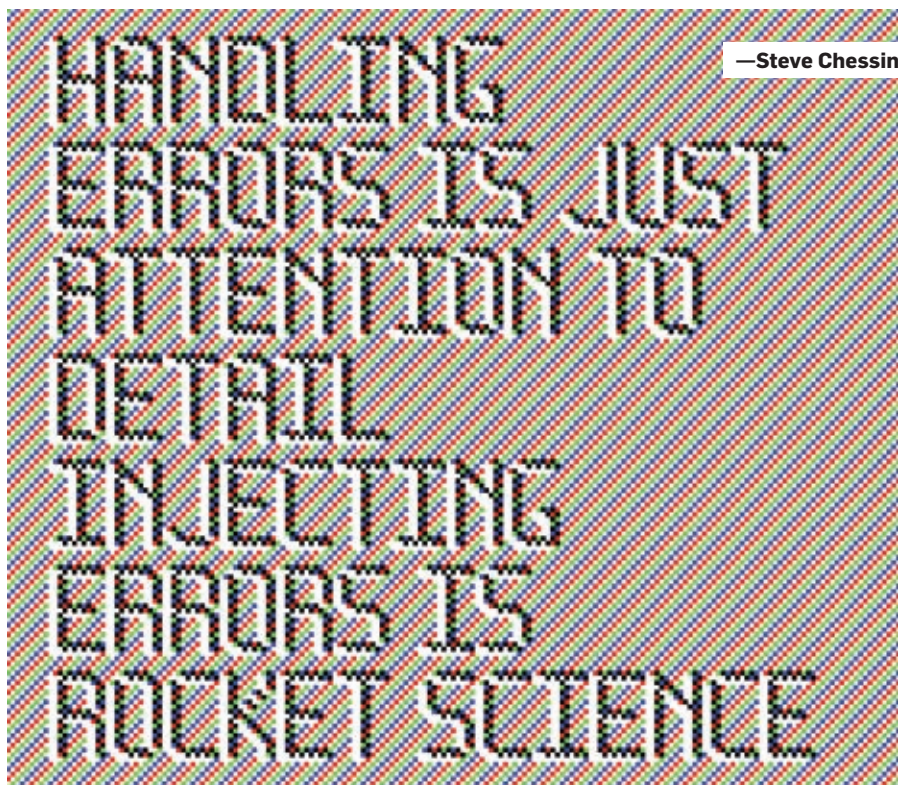
worse job if I were asked to design an ASIC). Instead, we based our error injector on one I had written in 1989 to test the memory parity error-recovery code I had written for Sun's SPARCstation-1. This error injector was modular and table-driven, and easily extensible. Of course, none of the actual low-level error-injection code applied to the UltraSPARC-II, so we hollowed it out and built upon the framework it provided.

The error injector consisted of two parts: a user-level command-line interface (`mtst`), and a device driver (`/dev/memtest`). The command-line interface allowed the user to specify whether the parity error should be injected onto a clean line or a dirty line and whether its detection should be triggered by a kernel load instruction, user-level load instruction, kernel instruction fetch, user-level instruction fetch, write-back to memory, snoop (copy-back) by another processor, or just left in a user-specified location in the cache. (This last was used by another user-level program, affectionately called the alphabomber, to measure the effectiveness of the cache scrubber.)

After parsing and processing its arguments, `mtst` would then open `/dev/memtest` and issue an `ioctl` to it. The parameters passed in the `ioctl` would tell the device driver whether to plant the error in its own space (for kernel-triggered errors) or at an address passed to it by `mtst` (for user-triggered errors) or at a specific cache location (for alphabombing). They would also specify if the device driver itself should trigger the error, and if so by a load instruction, an instruction fetch, a write-back to memory, or a copy-back to a different processor, and whether at trap-level zero or trap-level one. (For obvious reasons, neither `mtst` nor `/dev/memtest` are included in Solaris releases, nor is their source code included in OpenSolaris.)

Assuming the action of the device driver did not deliberately cause a kernel panic, it would return to `mtst`, which, depending upon the parameters with which it was invoked, would either trigger the error (by a load, instruction fetch, write-back, or snoop) or leave it in the cache (for alphabombing).

We later extended the error injector to produce timeouts and bus errors and to inject correctable and uncorrectable memory errors, so we eventu-



layers might. Thus, the error injector must be able to do all combinations.

Injecting E-Cache Errors on the UltraSPARC-II

"Handling errors is just attention to detail. Injecting errors is rocket science."

—me.

While the hardware engineers were working on determining the cause of the e-cache parity errors and then working on a fix, I was asked to lead a project to mitigate with software the effect of the errors. Unfortunately, the UltraSPARC-II used an imprecise trap to report e-cache parity errors detected

fashion (that is, one that would not cause a kernel panic) and flushed them from the cache before they could cause an outage. Whenever the system went idle, we flushed all clean lines, and all error-free dirty lines, from the cache.

Testing all of this required an error injector. While the hardware people had written one, it did not meet our needs; for example, you could only give it a physical address where the error was to be injected and wait for system code to trip over it. In addition, it was neither modular nor easily extensible (after all, it had been written by hardware people; to be fair, of course, I would do an even

ally had complete test coverage of all of the processor error-handling code in Solaris, something that had been lacking prior to this work. (The injection of correctable and uncorrectable memory errors is discussed later.)

The device driver used the diagnostic facilities of the UltraSPARC-II processor to inject the errors into the e-cache. (Similar diagnostic facilities were used by the cache scrubber.) Before I explain how that worked, it will help to understand the following:

- ▶ The UltraSPARC-II uses a 64-byte cache line.

- ▶ A cache line is moved between memory and the e-cache in 8-byte chunks.

- ▶ Each of these chunks is protected in memory by eight bits of ECC (error-correcting code) that can correct any single-bit error and detect any double-bit error (SEC-DED).

- ▶ Each byte of data is protected by a single parity bit when in the e-cache.

- ▶ There are two UDB (UltraSPARC Data Buffer) chips in parallel between the e-cache and main memory, and each UDB converts eight bytes of ECC-protected data at a time to eight bytes of parity-protected data (and vice versa). When a 64-byte cache line is moved from memory into the e-cache or vice versa, each UDB processes four 8-byte chunks.

The interface between the processor and the e-cache is 16 bytes wide. The processor's LSU (load/store unit) contains a control register that includes a 16-bit field called the force mask (FM). Each bit in the FM corresponds to one byte of the 16-byte interface between the CPU and the e-cache. When a bit is zero, a store of the corresponding byte is done with good parity. When a bit is one, a store of the corresponding byte is done with bad parity. The FM bits do not affect the checking of parity on loads from the e-cache.

Injecting a parity error into the e-cache is fairly straightforward. The physical memory address of the desired byte is determined, and the following steps performed:

1. Using its physical address, load the desired byte into a register; this has the side effect of bringing it into the e-cache if it isn't there already.

2. Disable interrupts.

3. Set LSU.FM to all ones.



As the density of circuits increases, features get smaller; as frequencies increase, voltages get lower. These trends combine to reduce the amount of charge used to represent a bit, increasing the sensitivity of memory to background radiation.



4. Store the desired byte back to its physical address. (If for some reason the containing cache line got displaced from the cache after the load, then this will bring it back into the cache.) The targeted byte will be written back into the cache line with bad parity.

5. Reset LSU.FM to zero.

6. Reenable interrupts.

Now that the desired byte is in the e-cache with bad parity, the latent error can be triggered via several mechanisms: data load in user or kernel mode, instruction fetch in user or kernel mode, displacement flush to cause a write-back, access from another CPU to cause a copy-back, and so on.

Interrupts must be disabled for the duration that the LSU.FM is not zero; otherwise, if an interrupt occurs and the interrupt handler (or any code it invokes) performs a store, then undesired parity errors will be introduced into the cache and triggered unpredictably.

This six-step sequence is used to inject e-cache parity errors at locations corresponding to specific physical memory addresses, kernel virtual addresses, or user virtual addresses. (Virtual addresses are translated to their corresponding physical addresses by the memtest device driver.) To simulate bit flips caused by background radiation, however, we would like to inject an e-cache parity error at an arbitrary e-cache offset, without regard to the physical memory address corresponding to the e-cache line.

Fortunately, the LSU.FM field also applies to stores to the e-cache using diagnostic accesses. Unfortunately, diagnostic loads and stores work only with 8-byte quantities, not with single bytes. In order to affect just a single byte, we must set only the one bit in LSU.FM that corresponds to the byte we want to change. The sequence in this case then becomes:

1. Disable interrupts.

2. Fool the instruction prefetcher (see below).

3. Set the desired bit in LSU.FM to one.

4. Load the containing eight bytes into a register with a diagnostic load.

5. Store the containing eight bytes back into the e-cache with a diagnostic store.

6. Reset LSU.FM to zero.

7. Reenable interrupts.

The only tricky part is preventing the contents of the e-cache from changing out from under us between the load and the store. The worst that snoop activity can do is change the state of a line from exclusive to shared, or from valid to invalid. As snooping cannot change the data itself, just the state in the tag, no harm is done if a snoop occurs between the load and the store.

However, there is one thing that can change the data in the cache between the load and the store. The processor contains an instruction prefetcher—one that is always on and whose behavior is not well documented in the *UltraSPARC I & II Users Manual*. The prefetcher is constantly moving instructions from the processor's i-cache (instruction cache) into the processor's instruction buffer. If the address of the next instruction to be prefetched misses in the i-cache, instructions will be brought in from the e-cache; if the address also misses in the e-cache, then the containing cache line will be brought into the e-cache from memory, displacing what was already there. If this e-cache fill happens to replace the line containing the byte we want to corrupt, and if the fill happens between the diagnostic load and the diagnostic store, we will write eight bytes of stale data into the e-cache (along with bad parity on one of them); this could cause an unexpected failure later if the line is reexecuted as an instruction. (Although we expect the byte with bad parity to cause an eventual failure, we want the failure to be the one we intended, not one we didn't intend.)

To prevent this, the prefetcher must be fooled into not prefetching for a while. Though this is possible—and in fact fairly easy to do—the procedure is not documented. The technique to use had to be obtained from the processor pipeline expert. In fact, if he hadn't informed us of this exposure, we would have had a hard-to-debug problem with the injector.

To fool the prefetcher, we statically position at the beginning of a cache line the code sequence that sets LSU.FM, issues the load and store, resets LSU.FM, reenables interrupts, and returns to the caller. When this routine is called, it disables interrupts and then branches just beyond the above sequence to a series of no-ops, enough

to fill the instruction buffer. The last instruction in this sequence branches back to the instruction that sets LSU.FM. Thus, when we get to the load of the load/store pair, the cache line that contains these instructions is already in the e-cache and has either already displaced the original target (so we will be injecting an error on top of our e-cache-resident code) or is in a different cache line than our target. In either case, the instruction prefetcher “sees” that the instructions (including the no-ops) that follow the load/store pair are already in the instruction buffer, so it temporarily has nothing to do. This prevents any lines from changing in the middle of the execution of the load/store pair. (This is the “rocket science” part of error injection.)

Of course, what would have really been nice would have been a control to turn off the instruction prefetcher.

Injecting Memory Errors on the UltraSPARC-II

“The horror of that moment,’ the King went on, ‘I shall never, NEVER forget!’ ‘You will, though,’ the Queen said, ‘if you don’t make a memorandum of it.’”

—Lewis Carroll,

Through the Looking Glass

Injecting memory errors on UltraSPARC-II systems is more difficult than injecting e-cache errors. As previously described, while the e-cache uses byte parity, memory uses eight bits of ECC to protect eight bytes. Data always moves between memory and the CPU subsystem (processor, two UDB chips, and e-cache) in 64-byte blocks, transferred in four 16-byte chunks. Each UDB handles eight bytes at a time, converting eight bytes with good ECC into eight bytes with good parity and vice versa.

Each UDB has a control register that contains an 8-bit FCBV (force check bit vector) field and an F_MODE (force mode) bit. When the F_MODE bit is set, the UDB uses the contents of the FCBV field for the ECC value on all outgoing (to memory) data, instead of calculating good ECC.

Since the FCBV field (when used) applies to all data going through the UDB, and since the smallest granule of transfer is 64 bytes, it is impossible to force bad ECC on just one arbitrary 8-byte extended word. (This means we cannot alphabomb CEs into arbitrary

locations.) Generating a single CE (correctable error) or UE (uncorrectable error) requires that the four 8-byte extended words passing through a given UDB start off as identical, so that they all share the same good ECC value.

Generating a CE or UE is typically done as follows:

1. Quiesce snoop activity, as snooped data goes through the UDBs.
2. Disable interrupts.
3. Set FCBV in the UDBs with the common good ECC value, and set their F_MODEs.
4. Load the desired 8-byte chunk into a register; this has the side effect of bringing it into the e-cache if it isn't there already.
5. Flip one (CE) or two (UE) bits in the register.
6. Store the now-modified 8-byte chunk; it will store into the cache and put the cache line into the modified state.
7. Displacement flush the cache line back to memory. The UDBs will convert each eight bytes with parity into eight bytes with ECC, but for the ECC bits they will use the value in the FCBV, which will be good for all but the modified chunk.
8. Clear F_MODE.
9. Enable interrupts.
10. Allow snoop activity.

(Although we could have confined the setting of FCBV and F_MODE to just the UDB handling the targeted location, it was easier to program them both identically.)

Snoop activity has to be quiesced; otherwise, any CPU or I/O device obtaining data out of this CPU's e-cache while the UDB's F_MODE bit is set will get bad ECC. Since I/O is difficult to quiesce, this is done by “pausing” all the other CPUs (by telling them to spin in a tight loop), and then flushing the cache so that the only owned line will be the one that we modify.

To inject a single CE at an arbitrary location, the UDB design should have included a “trigger” or “mask” field to indicate on which extended word(s) the FCBV field would be applied. This field could be, for example, an 8-bit mask, with one bit for each 8-byte chunk. (One UDB would use the even bits and the other would use the odd bits; this arrangement would make programming simpler.) The UDB would have to count the chunks going through it when the

F_MODE bit was set and apply FCBV to only those extended words that had the corresponding “trigger” bit(s) set.

Alternatively, the design could have included eight sets of FCBV fields (four in each UDB), each with its own F_MODE bit, so that arbitrary mixes of CEs, UEs, and good data could be planted at any location.

Other Uses of Diagnostic Access

“I’m running a Level 1 diagnostic.”

—Lt. Commander Geordi La Forge, in
Star Trek: The Next Generation

As illustrated earlier, diagnostic access to the e-cache and the memory interface chips is extremely important to

in the hardware. For example, while diagnostic access to the e-cache does not return the parity bits, the parity check logic works and sets the PSYND (parity syndrome) bits in the AFSR (Asynchronous Fault Status Register) as appropriate. (The 16 PSYND bits correspond to the 16 bytes in the interface between the processor and the e-cache. If a byte contains a parity error, the corresponding PSYND bit is set to one.) Thus, diagnostic access to the cache allows the parity bits to be inferred, if not observed directly.

It is important to note that the use of diagnostic access by the error injector and the cache scrubber depends

occurs the system board can send it to just those processors that contain copies of the snooped cache line, and not interfere with the performance of the processors that do not contain copies. Diagnostic access to DTAGs interferes with the maintenance of cache coherency, such that if an invalidate request came in at the same time as the diagnostic access, the invalidate request would be lost. (The request need not be for the particular line; all coherency traffic is ignored while the diagnostic request is being processed.)

This behavior makes it impossible to write a software DTAG scrubber, as the scrubber cannot determine if a line contains a latent error without risking the loss of system coherency.

Note that deciding whether to preserve coherency on a diagnostic access is an example of one of the many decisions a chip designer must make. Prior to Sun’s e-cache parity crisis, these decisions were made by the hardware designers without consulting the software error-handling experts. Since that crisis, error and diagnostic reviews of new chips are a required part of the hardware design cycle.

These reviews are joint meetings of the chip designers and the software people responsible for error handling, diagnosis, and containment. They are held early enough in the design process so that any deficiencies in the treatment of errors by the hardware (such as a failure to capture important information) can be corrected, and suggestions of improvements can be incorporated.

Other Methods of Error Injection

“Doctor, it hurts when I do this.”

“So don’t do that.”

—Henny Youngman

Hardware engineers have developed other methods for injecting errors, some more usable than others. For example, having learned from our e-cache parity experience, subsequent processors in the UltraSPARC line, beginning in about 2001 with the UltraSPARC-III, protect the e-cache with true ECC. In the UltraSPARC-III 16 bytes of data are protected by nine bits of ECC, and this same scheme is used to protect data in memory as well. (ECC is checked as data is moved from the e-cache to memory; single-bit errors are corrected and double-bit



error injection. Without the ability to use diagnostic loads and stores during normal system operation, injection of errors would be impossible.

Diagnostic access is also used in error prevention and correction, as the cache scrubber uses diagnostic loads to determine if a latent error is present, and to determine when lines should be displaced from the cache.

Diagnostic access is also used after a failure occurs, to read the contents of the affected cache line to aid in offline diagnosis. For this reason, it is important that diagnostic access provide visibility to all the bits, as they are stored

on their not interfering with normal system operation. In particular, system coherency must be maintained while the diagnostic operation is in progress.

The caches of UltraSPARC-II obey this requirement. Diagnostic access to the cache by the CPU does not interfere with the cache’s response to coherency traffic. Snooping continues, and requests to invalidate cache lines are processed normally.

Contrast this with the Sun Enterprise 10000 system board DTAG (dual tag), which contains a copy of the tag information in the four processors on the system board. Thus, when a snoop

errors are rewritten with a special syndrome. Similarly, ECC is checked as data is moved from memory to the e-cache; single-bit errors are corrected, but double-bit errors are written into the e-cache as is.)

Injecting memory errors in the UltraSPARC-III is similar to that in the UltraSPARC-II; a control register contains an FM bit and a forced ECC field. When set that ECC value is used instead of calculated ECC when data moves from the e-cache to memory.

For injecting errors into the UltraSPARC-III e-cache, the hardware engineers tried to do something similar; another control register contains an

registers: four to hold 32 bytes of data (a half-cache line consisting of two 16-byte ECC-protected chunks) and a fifth register to hold the two 9-bit ECC fields protecting the respective chunks. One set of diagnostic loads and stores moves data between the e-cache and the staging registers 32 data bytes and 18 ECC bits at a time; another set moves data between a given staging register and an integer register. This allows the error injector to flip any combination of data and ECC bits.

Conclusion

"Software. Hardware. Complete."

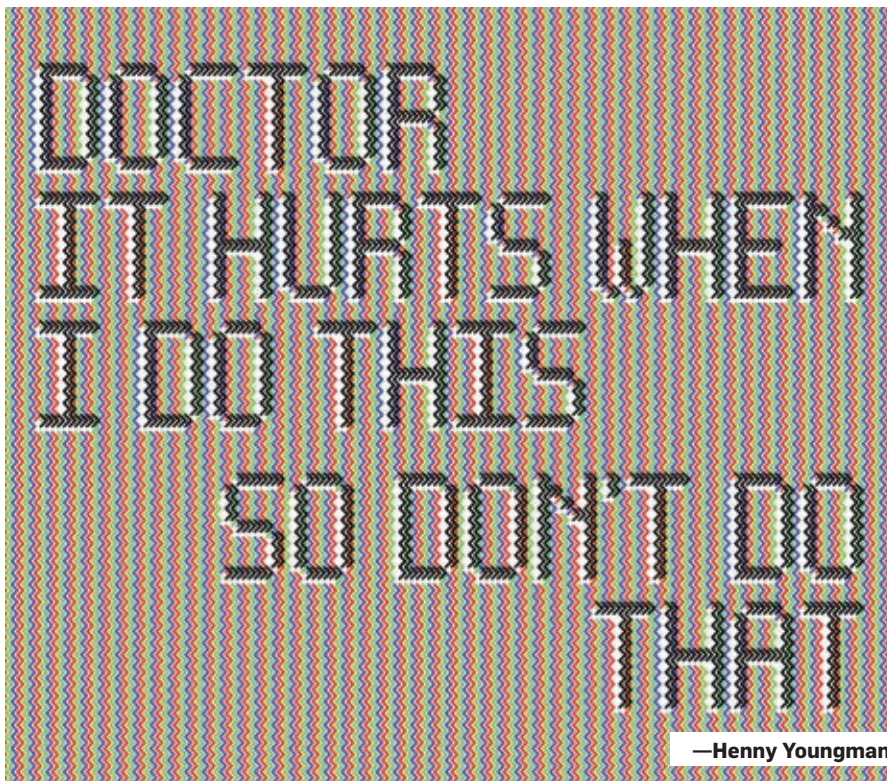
Since the e-cache parity crisis, error

advantage of error injection technology to improve the handling, diagnosis, and containment of errors by their respective systems, as having the hardware and software people all in a single organization allows the necessary continuous interaction between them as new hardware and software is developed. When hardware and software development is divided among different organizations, as it is in the Windows, VMware, and Linux worlds (or, alternatively, the Intel and AMD worlds), exploiting error injection technology for product improvement is much more difficult.

Acknowledgments

Much of this article is based on the work of the circa-1999 Solaris Software Recovery Project. That project was the result of cooperation between and hard work by many individuals from across Sun, including Mike Shapiro, Huay Yong Wang, Robert Berube, Jeff Bonwick, Michael Christensen, Mike Corcoran, John Falkenthal, Girish Goyal, Carl Gutekunst, Rajesh Harekal, Michael Hsieh, Tariq Magdon Ismail, Steven Lau, Patricia Levinson, Gavin Maltby, Tim Marsland, Richard McDougall, Allan McKillop, Jerriann Meyer, Scott Michael, Subhan Mohammed, Kevin Normoyle, Asa Romberger, Ashley Saulsbury, and Tarik Soydan. Robert Berube in particular did much of the initial coding of the UltraSPARC-II error injector.

I also want to thank Mike Shapiro and Jim Maurer for reviewing early drafts. Their suggestions have improved this article. Any errors that remain are solely my responsibility. **Q**



FM bit and a forced ECC field, only the forced ECC in this register is used whenever data is written into the e-cache. This would have been difficult to use, as stores do not write data directly into the e-cache, but into a w-cache (write cache). The data in the w-cache is not merged with that in the e-cache until the line is displaced out of the w-cache, and that is difficult to control. Fortunately, we did not have to use this mechanism, as the hardware engineers provided something even better: direct access to the raw bits in the e-cache, both data and ECC.

This mechanism uses five staging

injection has become a core competency of what is now Oracle's Systems organization. As new processors and their supporting ASICs are designed, error and diagnostic reviews make sure they have the appropriate ability to inject errors into their internal structures, and the error injector is enhanced to inject those errors so that we can test our error-handling, diagnosis, and containment software in an end-to-end fashion.

Of course, companies such as IBM and Oracle that control both the hardware they sell and the software that supports it are best positioned to take

Related articles on queue.acm.org

Self-Healing in Modern Operating Systems

Michael W. Shapiro

<http://queue.acm.org/detail.cfm?id=1039537>

A Conversation with Jeff Bonwick and Bill Moore

<http://queue.acm.org/detail.cfm?id=1317400>

You Don't Know Jack about Disks

Dave Anderson

<http://queue.acm.org/detail.cfm?id=864058>

Steve Chessin (steve.chessin@oracle.com) is a senior principal software engineer in the Systems Group Quality organization of Oracle Corporation, Menlo Park, CA.

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.



Improving the performance of complex software is difficult, but understanding some fundamental principles can make it easier.

BY CARY MILLSAP

Thinking Clearly About Performance, Part 1

WHEN I JOINED Oracle Corporation in 1989, performance—what everyone called “Oracle tuning”—was difficult. Only a few people claimed they could do it very well, and those people commanded high consulting rates. When circumstances thrust me into the “Oracle tuning” arena, I was quite unprepared.

Recently, I’ve been introduced to the world of “MySQL tuning,” and the situation seems very similar to what I saw in Oracle more than 20 years ago.

It reminds me a lot of how difficult beginning algebra seemed when I was about 13 years old. At that age, I had to appeal heavily to trial and error to get through. I can remember looking at an equation such as $3x + 4 = 13$ and basically stumbling upon the answer, $x = 3$.

The trial-and-error method worked—albeit slowly and uncomfortably—for easy equations, but it didn’t scale as the problems got tougher—for example, $3x + 4 = 14$. Now what? My

problem was that I wasn’t thinking clearly yet about algebra. My introduction at age 15 to teacher James R. Harkey put me on the road to solving that problem.

In high school Mr. Harkey taught us what he called an *axiomatic* approach to solving algebraic equations. He showed us a set of steps that worked every time (and he gave us plenty of homework to practice on). In addition, by executing those steps, we necessarily *documented* our thinking as we worked. Not only were we thinking clearly, using a reliable and repeatable sequence of steps, but we were also *proving* to anyone who read our

work that we were thinking clearly. Our work for Mr. Harkey is illustrated in Table 1.

This was Mr. Harkey's axiomatic approach to algebra, geometry, trigonometry, and calculus: one small, logical, provable, and auditable step at a time. It's the first time I ever really got mathematics.

Naturally, I didn't realize it at the time, but of course *proving* was a skill that would be vital for my success in the world after school. In life I've found that, of course, knowing things matters, but *proving* those things to other people matters more. Without good *proving* skills, it's difficult to be a good consultant, a good leader, or even a good employee.

My goal since the mid-1990s has been to create a similarly rigorous approach to Oracle performance optimization. Lately, I have been expanding the scope of that goal beyond Oracle to: "Create an axiomatic approach to computer software performance optimization." I've found that not many people like it when I talk like that, so let's say it like this: "My goal is to help you think clearly about how to optimize the performance of your computer software."

What is Performance?

Googling the word *performance* results in more than a half-billion hits on concepts ranging from bicycle racing to the dreaded employee review process that many companies these days are learning to avoid. Most of the top hits relate to the subject of this article: the time it takes for computer software to perform whatever task you ask it to do.

And that's a great place to begin: the task, a business-oriented unit of work. Tasks can nest: "print invoices" is a task; "print one invoice"—a sub-task—is also a task. For a computer

user, *performance* usually means the time it takes for the system to execute some task. *Response time* is the execution duration of a task, measured in time per task, such as "seconds per click." For example, my Google search for the word *performance* had a response time of 0.24 seconds. The Google Web page rendered that measurement right in my browser. That is evidence to me that Google values my perception of Google performance.

Some people are interested in another performance measure: *throughput*, the count of task executions that complete within a specified time interval, such as "clicks per second." In general, people who are responsible for the performance of *groups* of people worry more about throughput than does the person who works in a solo contributor role. For example, an individual accountant is usually more concerned about whether the response time of a daily report will require that accountant to stay late after work. The manager of a group of accounts is additionally concerned about whether the system is capable of processing all the data that all of the accountants in that group will be processing.

Response Time versus Throughput

Throughput and response time have a generally reciprocal type of relationship, but not exactly. The real relationship is subtly complex.

Example 1. *Imagine that you have measured your throughput at 1,000 tasks per second for some benchmark. What, then, is your users' average response time? It's tempting to say that the average response time is $1/1,000 = .001$ seconds per task, but it's not necessarily so.*

Imagine that the system processing this throughput had 1,000 parallel, independent, homogeneous service chan-

nels (that is, it's a system with 1,000 independent, equally competent service providers, each awaiting your request for service). In this case, it is possible that each request consumed exactly 1 second.

Now, you can know that average response time was somewhere between 0 and 1 second per task. You cannot derive response time exclusively from a throughput measurement, however; you have to measure it separately (I carefully include the word exclusively in this statement, because there are mathematical models that can compute response time for a given throughput, but the models require more input than just throughput).

The subtlety works in the other direction, too. You can certainly flip this example around and prove it. A scarier example, however, will be more fun.

Example 2. *Your client requires a new task that you're programming to deliver a throughput of 100 tasks per second on a single-CPU computer. Imagine that the new task you've written executes in just .001 seconds on the client's system. Will your program yield the throughput the client requires?*

It's tempting to say that if you can run the task once in just one thousandth of a second, then surely you'll be able to run that task at least 100 times in the span of a full second. And you're right, if the task requests are nicely serialized, for example, so that your program can process all 100 of the client's required task executions inside a loop, one after the other.

But what if the 100 tasks per second come at your system at random, from 100 different users logged into your client's single-CPU computer? Then the gruesome realities of CPU schedulers and serialized resources (such as Oracle latches and locks and writable access to buffers in memory) may restrict your throughput to quantities much less than the required 100 tasks per second. It might work; it might not. You cannot derive throughput exclusively from a response time measurement. You have to measure it separately.

Response time and throughput are not necessarily reciprocals. To know them both, you need to measure them both. Which is more important? For a given situation, you might answer legitimately in either direction. In

Table 1. The axiomatic approach as taught by Mr. Harkey.

$3.1x + 4$	$= 13$	problem statement
$3.1x + 4 - 4$	$= 13 - 4$	subtraction property of equality
$3.1x$	$= 9$	additive inverse property, simplification
$3.1x / 3.1$	$= 9 / 3.1$	division property of equality
x	≈ 2.903	multiplicative inverse property, simplification

many circumstances, the answer is that both are vital measurements requiring management. For example, a system owner may have a business requirement not only that response time must be 1.0 second or less for a given task in 99% or more of executions but also that the system must support a sustained throughput of 1,000 executions of the task within a 10-minute interval.

Percentile Specifications

Earlier, I used the phrase “in 99% or more of executions” to qualify a response time expectation. Many people are more accustomed to such statements as “average response time must be r seconds or less.” The percentile way of stating requirements maps better, though, to the human experience.

Example 3. *Imagine that your response time tolerance is 1 second for some task that you execute on your computer every day. Imagine further that the lists of numbers shown in Table 2 represent the measured response times of 10 executions of that task. The average response time for each list is 1.000 second. Which one do you think you would like better?*

Although the two lists in Table 2 have the same average response time, the lists are quite different in character. In list A, 90% of response times were one second or less. In list B, only 60% of response times were one second or less. Stated in the opposite way, list B represents a set of user experiences of which 40% were dissatisfactory, but list A (having the same average response time as list B) represents only a 10% dissatisfaction rate.

In list A, the 90th percentile response time is .987 seconds; in list B, it is 1.273 seconds. These statements about percentiles are more informative than merely saying that each list represents an average response time of 1.000 second.

As GE says, “Our customers feel the variance, not the mean.”¹ Expressing response-time goals as percentiles makes for much more compelling requirement specifications that match with end-user expectations: for example, the “Track Shipment” task must complete in less than .5 seconds in at least 99.9% of executions.

Problem Diagnosis

In nearly every performance problem I’ve been invited to repair, the stated problem has been about response time: “It used to take less than a second to do X; now it sometimes takes 20+.” Of course, a specific statement like that is often buried under veneers of other problems such as: “Our whole [adjectives deleted] system is so slow we can’t use it.”²

Just because something happened often for me doesn’t mean it will happen for you. The most important thing to do first is state the problem clearly, so you can think about it clearly.

A good way to begin is to ask, what is the goal state that you want to achieve? Find some specifics that you can *measure* to express this: for example, “Response time of X is more than 20 seconds in many cases. We’ll be happy when response time is one second or less in at least 95% of executions.” That sounds good in theory, but what if your user doesn’t have such a specific quantitative goal? This particular goal has two quantities (1 and 95); what if your user doesn’t know either one of them? Worse yet, what if your user *does* have specific ideas, but those expectations are impossible to meet? How would you know what

“possible” or “impossible” even is?

Let’s work our way through those questions.

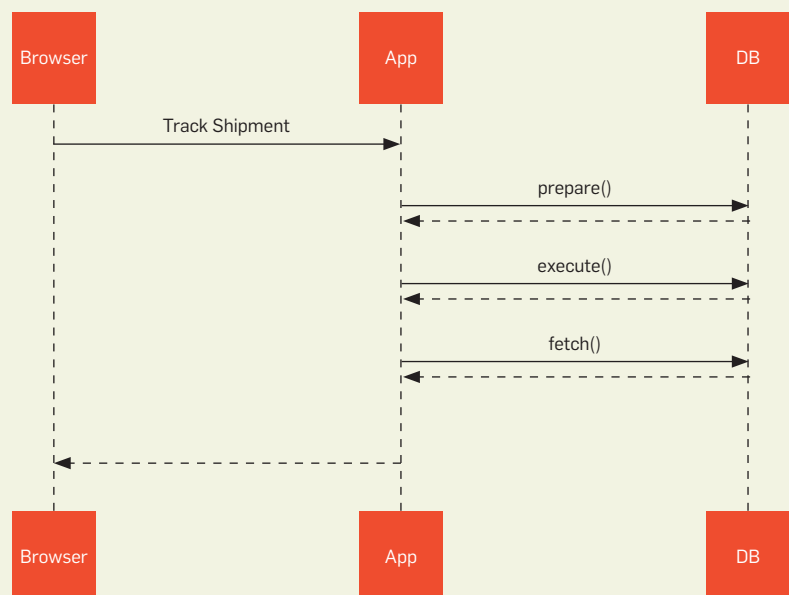
The Sequence Diagram

A *sequence diagram* is a type of graph specified in UML (Unified Modeling Language), used to show the interactions between objects in the sequential order that those interactions occur. The sequence diagram is an exceptionally useful tool for visualizing response time. Figure 1 shows a standard UML sequence diagram for a simple application system composed

Table 2. The average response time for each of these two lists is 1.000 second.

	List A	List B
1	.924	.796
2	.928	.798
3	.954	.802
4	.957	.823
5	.961	.919
6	.965	.977
7	.972	1.076
8	.979	1.216
9	.987	1.273
10	1.373	1.320

Figure 1. This UML sequence diagram shows the interactions among a browser, an application server, and a database.



of a browser, application server, and a database.

Imagine now drawing the sequence diagram to scale, so that the distance between each “request” arrow coming

in and its corresponding “response” arrow going out are proportional to the duration spent servicing the request. I have shown such a diagram in Figure 2. This is a good graphical

representation of how the components represented in your diagram are spending your user’s time. You can “feel” the relative contribution to response time by looking at the picture.

Sequence diagrams are just right for helping people conceptualize how their responses are consumed on a given system, as one tier hands control of the task to the next. Sequence diagrams also work well to show how simultaneous processing threads work in parallel, and they are good tools for analyzing performance outside of the information technology business.¹

The sequence diagram is a good conceptual tool for talking about performance, but to think clearly about performance, you need something else. Here’s the problem. Imagine the task you’re supposed to fix has a response time of 2,468 seconds (41 minutes, 8 seconds). In that period of time, running that task causes your application server to execute 322,968 database calls. Figure 3 shows what the sequence diagram for that task would look like.

There are so many request and response arrows between the application and database tiers that you can’t see any of the detail. Printing the sequence diagram on a very long scroll isn’t a useful solution, because it would take weeks of visual inspection before you would be able to derive useful information from the details you would see.

The sequence diagram is a good tool for conceptualizing flow of control and the corresponding flow of time. To think clearly about response time, however, you need something else.

The Profile

The sequence diagram does not scale well. To deal with tasks that have huge call counts, you need a convenient aggregation of the sequence diagram so that you understand the most important patterns in how your time has been spent. Table 3 shows an example of a *profile*, which does the trick. A profile is a tabular decomposition of response time, typically listed in descending order of component response time contribution.

Example 4. The profile in Table 3

Figure 2. A UML sequence diagram drawn to scale, showing the response time consumed at each tier in the system.

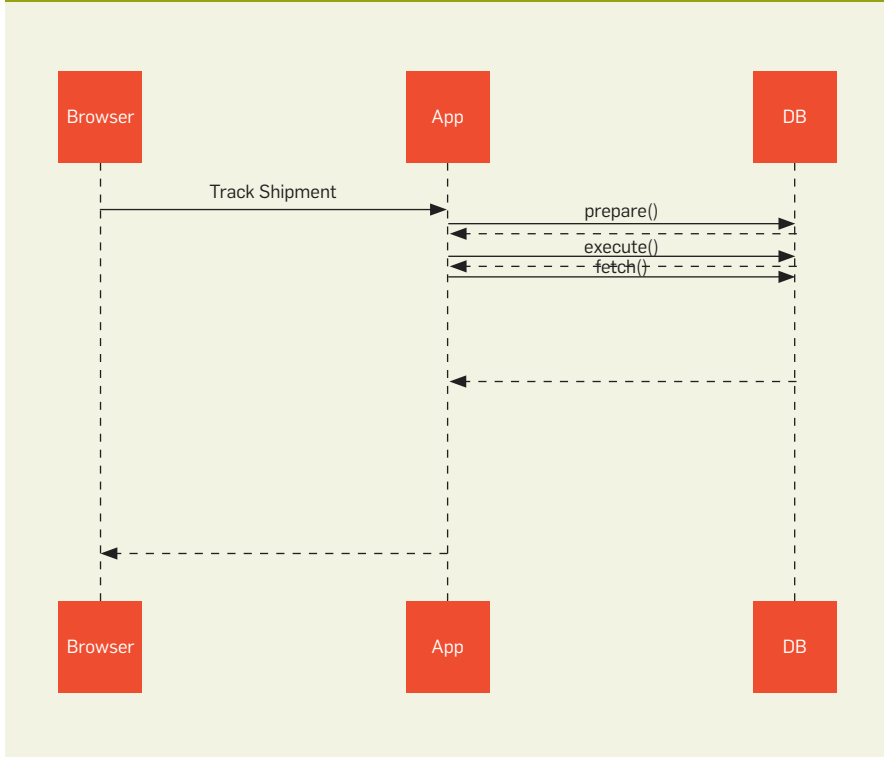
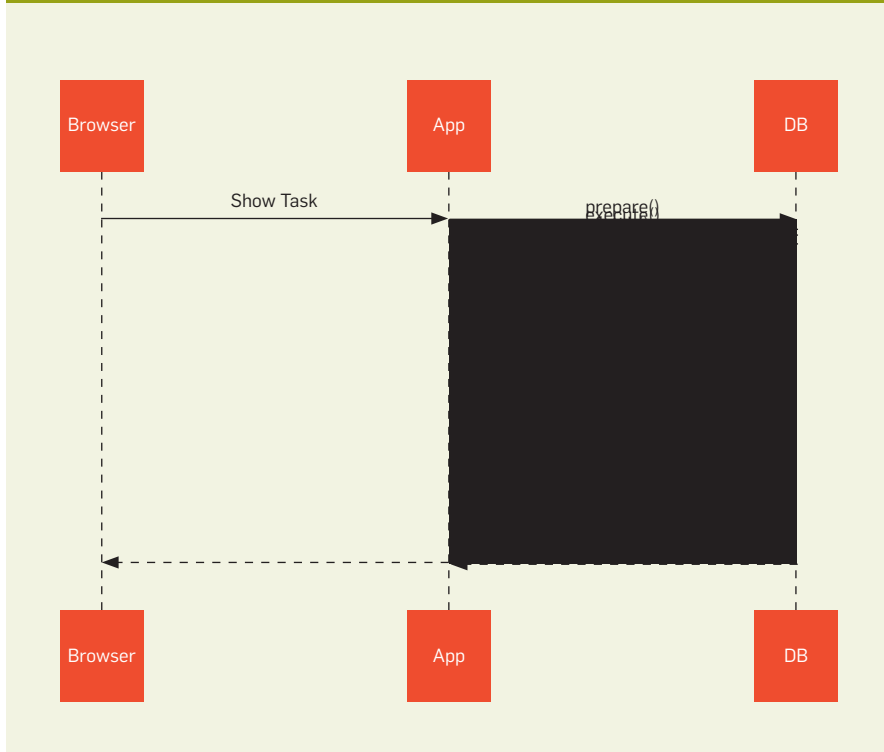


Figure 3. This UML sequence diagram shows 322,968 database calls executed by the application server.



is rudimentary, but it shows exactly where your slow task has spent your user's 2,468 seconds. With the data shown here, for example, you can derive the percentage of response time contribution for each of the function calls identified in the profile. You can also derive the average response time for each type of function call during your task.

A profile shows where your code has spent your time and—sometimes even more importantly—where it has not. There is tremendous value in not having to guess about these things.

From the data shown in Table 3, you know that 70.8% of your user's response time is consumed by `DB:fetch()` calls. Furthermore, if you can drill down in to the individual calls whose durations were aggregated to create this profile, you can know how many of those `App:await_db_netIO()` calls corresponded to `DB:fetch()` calls, and you can know how much response time each of those consumed. With a profile, you can begin to formulate the answer to the question, "How long should this task run?"... which, by now, you know is an important question in the first step (section 0) of any good problem diagnosis.

Amdahl's Law

Profiling helps you think clearly about performance. Even if Gene Amdahl had not given us Amdahl's Law back in 1967, you would probably have come up with it yourself after the first few profiles you looked at.

Amdahl's Law states: Performance improvement is proportional to how much a program uses the thing you improved. If the thing you're trying to improve contributes only 5% to your task's total response time, then the maximum impact you'll be able to make is 5% of your total response time. This means that the closer to the top of a profile that you work (assuming that the profile is sorted in descending response-time order), the bigger the benefit potential for your overall response time.

This doesn't mean that you always work a profile in top-down order, though, because you also need to consider the *cost* of the remedies you'll be executing.³

Example 5. Consider the profile in

Table 4. It's the same profile as in Table 3, except here you can see how much time you think you can save by implementing the best remedy for each row in the profile, and you can see how much you think each remedy will cost to implement.

Which remedy action would you implement first? Amdahl's Law says that implementing the repair on line 1 has the greatest potential benefit of saving about 851 seconds (34.5% of 2,468 seconds). If it is truly "super expensive," however, then the remedy on line 2 may yield better a net benefit—and that is the constraint to which you really need to optimize—even though the potential for response time savings is only about 305 seconds.

A tremendous value of the profile is that you can learn exactly how much improvement you should expect for a proposed investment. It opens the door to making much better decisions about what remedies to implement first. Your predictions give you

a yardstick for measuring your own performance as an analyst. Finally, it gives you a chance to showcase your cleverness and intimacy with your technology as you find more efficient remedies for reducing response time more than expected, at lower-than-expected costs.

What remedy action you implement first really boils down to how much you trust your cost estimates. Does "dirt cheap" really take into account the risks that the proposed improvement may inflict upon the system? For example, it may seem dirt cheap to change that parameter or drop that index, but does that change potentially disrupt the good performance behavior of something out there that you're not even thinking about right now? Reliable cost estimation is another area in which your technological skills pay off.

Another factor worth considering is the political capital that you can earn by creating small victories. Maybe

Table 3. This profile shows the decomposition of a 2,468.000-second response time.

Function Call	R (sec)	Calls
1 DB: fetch()	1,748.229	322,968
2 App: await_db_netIO()	338.470	322,968
3 DB: execute()	152.654	39,142
4 DB: prepare()	97.855	39,142
5 Other	58.147	89,422
6 App: render_graph()	48.274	7
7 App: tabularize()	23.481	4
8 App: read()	0.890	2
Total	2,468.000	

Table 4. This profile shows the potential for improvement and the corresponding cost (difficulty) of improvement for each line item from Table 2.

Potential improvement % and cost of investment	R (sec)	R (%)
1 34.5% super expensive	1,748.229	70.8%
2 12.3% dirt cheap	338.470	13.7%
3 Impossible to improve	152.654	6.2%
4 4.0% dirt cheap	97.855	4.0%
5 0.1% super expensive	58.147	2.4%
6 1.6% dirt cheap	48.274	2.0%
7 Impossible to improve	23.481	1.0%
8 0.0% dirt cheap	0.890	0.0%
Total	2,468.000	

Table 5. A skew histogram for the 322,968 calls from Table 2.

	Range [min ≤ e < max]		R (sec)	Calls
1	0	0.000001	0.000	0
2	0.000001	0.00001	0.002	397
3	0.00001	0.0001	0.141	2,169
4	0.0001	0.001	31.654	92,557
5	0.001	0.01	389.662	180,399
6	0.01	0.1	1,325.870	47,444
7	0.1	1	0.900	2
Total			1,748.229	322,968

cheap, low-risk improvements won't amount to much overall response-time improvement, but there's value in establishing a track record of small improvements that exactly fulfill your predictions about how much response time you'll save for the slow task. A track record of prediction and fulfillment ultimately—especially in the area of software performance, where myth and superstition have reigned at many locations for decades—gives you the credibility you need to influence your colleagues (your peers, your managers, your customers...) to let you perform increasingly expensive remedies that may produce bigger payoffs for the business.

A word of caution, however: don't get careless as you rack up successes and propose ever-bigger, costlier, riskier remedies. Credibility is fragile. It takes a lot of work to build it up but only one careless mistake to bring it down.

Skew

When you work with profiles, you repeatedly encounter sub-problems such as this:

Example 6. *The profile in Table 3 revealed that 322,968 DB: fetch() calls had consumed 1,748.229 seconds of response time. How much unwanted response time would be eliminated if you could eliminate half of those calls? The answer is almost never, "Half of the response time." Consider this far simpler example for a moment:*

Example 7. *Four calls to a subroutine consumed four seconds. How much unwanted response time would be eliminated if you could eliminate half of those calls? The answer depends upon*

the response times of the individual calls that we could eliminate. You might have assumed that each of the call durations was the average $4/4 = 1$ second, but nowhere did the statement tell you that the call durations were uniform.

Imagine the following two possibilities, where each list represents the response times of the four subroutine calls:

A = {1, 1, 1, 1}
B = {3.7, .1, .1, .1}

In list A, the response times are uniform, so no matter which half (two) of the calls you eliminate, you will reduce total response time to two seconds. In list B, however, it makes a tremendous difference which two calls are eliminated. If you eliminate the first two calls, then the total response time will drop to .2 seconds (a 95% reduction). If you eliminate the final two calls, then the total response time will drop to 3.8 seconds (only a 5% reduction).

Skew is a nonuniformity in a list of values. The possibility of skew is what prohibits you from providing a precise answer to the question I asked at the beginning of this section. Let's look again:

Example 8. *The profile in Table 3 revealed that 322,968 DB: fetch() calls had consumed 1,748.229 seconds of response time. How much unwanted response time would you eliminate by eliminating half of those calls? Without knowing anything about skew, the most precise answer you can provide is, "Somewhere between 0 and 1,748.229 seconds."*

Imagine, however, that you had the additional information available

in Table 5. Then you could formulate much more precise best-case and worst-case estimates. Specifically, if you had information like this, you would be smart to try to figure out how specifically to eliminate the 47,444 calls with response times in the .01- to .1-second range.

Summary

In Part 1, I have tried to link together some of the basic principles that I have seen people trip over in my travels as a software performance analyst. In Part 2, I will describe how competition for shared resources influences performance by covering the concepts of *efficiency*, *load*, *queuing delay*, and *coherency delay*. I will also explain how to think clearly about performance during the design, build, and test phases of an application, so that you'll be much more likely to create fast software that can become even faster throughout its production lifespan. 

Related articles on queue.acm.org

You're Doing It Wrong

Poul-Henning Kamp

<http://queue.acm.org/detail.cfm?id=1814327>

Performance Anti-Patterns

Bart Smalders

<http://queue.acm.org/detail.cfm?id=1117403>

Hidden in Plain Sight

Bryan Cantrill

<http://queue.acm.org/detail.cfm?id=1117401>

References

1. General Electric Company. What is Six Sigma? The roadmap to customer impact; <http://www.ge.com/sixsigma/SixSigma.pdf>.
2. Millsap, C. My whole system is slow. Now what? 2009; <http://carymillsap.blogspot.com/2009/12/my-whole-system-is-slow-now-what.html>.
3. Millsap, C. On the importance of diagnosing before resolving. 2009; <http://carymillsap.blogspot.com/2009/09/on-importance-of-diagnosing-before.html>.
4. Millsap, C. Performance optimization with Global Entry. Or not? 2009; <http://carymillsap.blogspot.com/2009/11/performance-optimization-with-global.html>.

Cary Millsap is the founder and president of Method R Corporation (<http://method-r.com>), a company devoted to software performance. He is the author (with Jeff Holt) of *Optimizing Oracle Performance* (O'Reilly) and a co-author of *Oracle Insights: Tales of the Oak Table* (Apress). He is the former vice president of Oracle Corporation's System Performance Group and is also an Oracle ACE Director and a founding partner of the Oak Table Network, an informal association of well-known "Oracle scientists." He blogs at <http://carymillsap.blogspot.com>, and he tweets at <http://twitter.com/CaryMillsap>.

© 2010 ACM 0001-0782/10/0900 \$10.00



Group Term Life Insurance

10- or 20-Year Group Term Life Insurance

Group Disability Income Insurance

Group Accidental Death & Dismemberment Insurance

Group Catastrophic Major Medical Insurance

Group Dental Plan

Long-Term Care Plan

Major Medical Insurance

Short-Term Medical Plan

Who has time to think about insurance?

Today, it's likely you're busier than ever. So, the last thing you probably have on your mind is whether or not you are properly insured.

But in about the same time it takes to enjoy a cup of coffee, you can learn more about your ACM-sponsored group insurance program — a special member benefit that can help provide you financial security at economical group rates.

Take just a few minutes today to make sure you're properly insured.

Call Marsh U.S. Consumer, a service of Seabury & Smith, Inc., at 1-800-503-9230 or visit www.personal-plans.com/promo/acm/49771.

49771 (2010) ©Seabury & Smith, Inc. 2010

Administered by:

MARSH



MARSH MERCER KROLL
GUY CARPENTER OLIVER WYMAN

d/b/a in CA Seabury & Smith Insurance Program Management
CA Ins. Lic. #0633005
AR Ins. Lic. #245544

DOI:10.1145/1810891.1810911

Computing research ages more slowly than research in other scientific disciplines, supporting the call for parity in funding.

BY DAG I.K. SJØBERG

Confronting the Myth of Rapid Obsolescence in Computing Research

COMPUTING TECHNOLOGIES ARE changing everyone's social, political, economic, and cultural worlds.¹² Meanwhile, scientists commonly believe that research in computing is advancing more quickly and just as quickly becoming obsolete more quickly than research in other scientific disciplines. A notable indicator is how quickly it stops being cited in the literature. Common measures of this phenomenon are called "cited half-life," "citing half-life," and the Price Index (see the sidebar "Definitions and Measures of Obsolescence"). These measures show that research

in computing does not cease being cited any more quickly than research in other disciplines, indicating (contrary to popular belief) that research in computing does not become obsolete more quickly than research in other disciplines. The extent to which this is the case is important for several reasons:

Demand for innovation. Though computing has made great strides, society continues to demand more complex, reliable, robust, usable hardware and software systems. Advances in computing technology needed to meet it depend on long-term funding of fundamental research.¹¹ However, it can be difficult to convince funding bodies to support long-term fundamental research programs in computing. One reason may be the already quick pace of development of computing applications, perhaps suggesting that the research is not as difficult as in other disciplines and that progress can be made with less funding than other disciplines. Hence, as has been reported in the context of U.S. National Science Foundation research-funding policy, when competing for research money, computer scientists argue that society has a compelling need for the results of their research, as well as CS as a basic research discipline to maintain its standing within the scientific community.¹⁹ Competing for funding with researchers from other sciences in a university setting, CS researchers must counter the argument that research funding in computing will not be prioritized because everything useful is already being done both faster and better by the IT industry anyway.

» key insights

- With respect to aging in the research literature, CS is in the middle of the scientific disciplines.
- The research challenges in computing are as fundamental and long-lasting as those in other disciplines.
- Publication delay is not a major problem within CS.



Aging research. Though relevant, the CS literature may still be considered obsolete and thereby ignored due to its age. As a researcher and journal editor, I find that reviewers frequently mention “old references,” and, as a supervisor, I find Ph.D. students are often reluctant to read older literature.

Publication delay. Researchers in computing sometimes claim the relatively long lag between submission and publication of a journal article renders the research outdated before publication, arguing for submitting their manuscripts to conferences rather than to journals.

Library ROI. Due to the ever-increasing volume of research literature, libraries must make cost-effective decisions, identifying the core journals within a discipline, canceling their subscriptions to less-accessed jour-

nals, and archiving less-accessed material to save shelf space. To maximize return on their investment, libraries must collect statistics on the use of their materials.⁹ Research literature on computing being accessed less often or quickly becoming obsolete may affect decisions about the archiving and retention of computing journals.

Results

Table 1 reflects CS within the various disciplines with respect to average aggregated cited, citing half-lives, and Price Index. This result is in striking contrast to the only other work I found on obsolescence of the computing literature—Cunningham and Bocock³—which found a citing half-life of four years (I found 7.5), concluding that their study supported “...a commonly held belief about computer science,

that it is a rapidly changing field with a relatively high obsolescence rate for its documents. This hypothesis is confirmed for the field of computer operating systems and network management...” They also reported a half-life of five years for the field of “Information Systems.” The main reason for the discrepancy between their results and mine is likely that they based their analysis on a small sample—only two journals (one that no longer exists) and four issues of the proceedings of one conference, the International Conference on Information Systems. By contrast, *ISI Journal Citation Report (JCR)* provided me with values for 382 computing journals.

The extent to which the cited and citing half-life measures are equivalent or complementary has been covered in the literature.¹⁷ For individual

Definitions and Measures of Obsolescence

All disciplines include foundational research that is relevant to ongoing research but that does not need to be (and is not) cited; an example is Newton's *Principia Mathematica*. However, it is generally reasonable to assume that researchers cite other research in their publications if they consider it relevant. Consequently, the median age of the references to, or in, published articles within a field is an indicator of how quickly the literature becomes obsolete. Inspired by the same term in nuclear physics, this indicator, called "half-life," has been used for a long time in bibliometric research.¹⁸

There are basically two ways—retrospective studies and prospective studies²—to weigh the obsolescence of an article, a journal, or the body of the literature of a (sub)field. In retrospective studies, one proceeds backward from a particular date. The JCR⁸ provides two retrospective half-life measures:

Cited half-life. The cited half-life of a journal for a particular year is the number of years (counting backward from and including that year) accounting for 50% of the citations received from the sample of journals under consideration. Cited half-life shows how quickly articles published in a given journal, taken as a set, cease to be cited. To illustrate, the red arrows in the figure here indicate articles published in 2007 in various journals that cite articles in *Communications* independent of year; assume in this example only one article per journal. One must then go back to 1998 to include 50% of the citations to *Communications*, giving a cited half-life of 10 years. In reality, JCR listed 8,969 citations from articles published in 2007 in 624 journals or other sources to articles published in *Communications*, but 63% of the citations were to articles published in 1997 or earlier; that is, the cited half-life was greater than 10 years. (For half-lives >10, JCR reports only the text ">10.") The definition of cited half-life can be modified to cover subject categories or research fields by considering citations to articles in a set of journals representing the category or field. This aggregate cited half-life is an indicator of the turnover rate of the body of work on a subject or in a field.

Citing half-life. The citing half-life for a particular year is the median age of all articles cited in a given sample of articles. In the figure, the blue arrows indicate citations in 2007 *Communications* articles to five articles in various journals in different years. The median year of publication of these five articles is 2003; the citing half-life is five years. In reality, the citing half-life for *Communications* in 2007, as reported by JCR, was 5.5 years, calculated on the basis of 1,607 citations to articles in 155 journals or

other sources. (JCR reported half-life values in decimals because it used an interpolation formula in the calculation.) The citing half-life shows how quickly a journal ceases citing articles from itself or from other sources. This definition can be modified to cover subject categories or research fields.

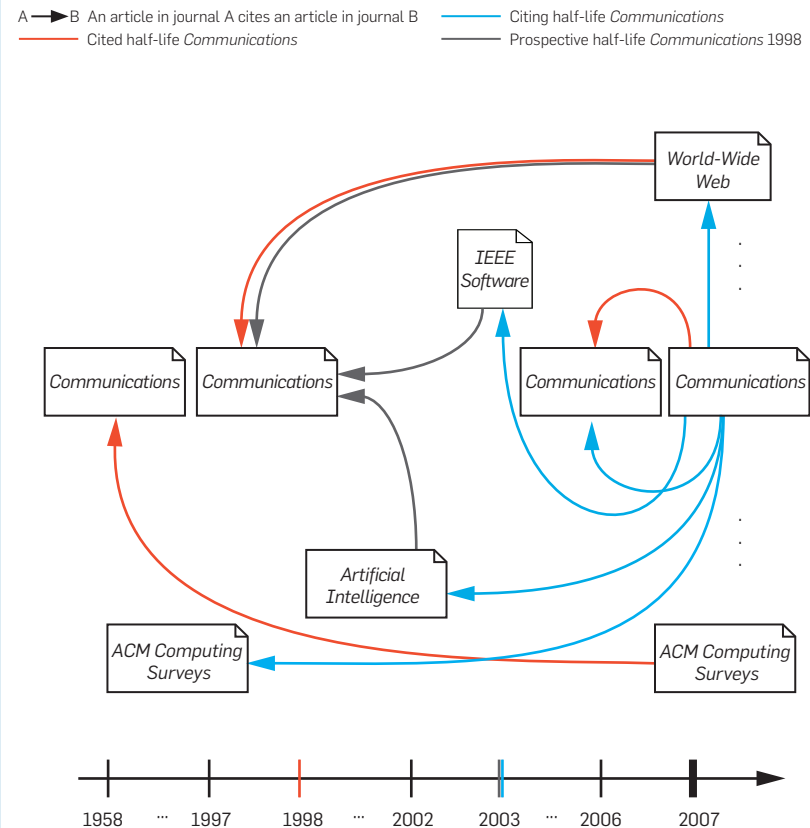
Related to the measure of citing half-life is the Price Index, defined as the proportion of articles cited by a publication that is no more than five years older than the publication doing the citing.⁴ The index is an outcome of Derek J. de Solla Price's work at the University of Malaya in Singapore, Cambridge University, and Yale University in the 1950s and 1960s on the growth of knowledge in science and the "research front of recent papers."⁵ A large index value indicates a discipline characterized by quick growth and an active research front.

In prospective studies, one investigates the history of citations that have been made to a particular article or set of articles after publication over

a given time period, typically 10 to 15 years.⁶ Half-life is defined as the time period over which half the citations to the (set of) articles were made; for example, if we were to calculate the prospective half-life of *Communications* for 1998 in a 10-year window, we would have to determine the number of citations to the 1998 *Communications* articles from 1998 to 2007. The figure outlines how articles published in the journals *Artificial Intelligence* in 2002, *IEEE Software* in 2003, and *World Wide Web* in 2007 cited articles in *Communications* in 1998 (dark gray arrows). If the citations were from one article in each of the three journals, the prospective half-life would be six years, and the median citation would be in *IEEE Software*.

The advantage of prospective half-lives is that researchers are able to track the use of individual articles. However, calculating prospective half-lives is challenging and not provided by JCR. Consequently, I don't report prospective half-lives here but include the definition for the sake of completeness.

Citation half-lives.



journals, the values may be quite different (such as the respective values >10 and 5.5 for *Communications*). Nevertheless, as in Table 1, there is strong correlation among the three measures of obsolescence at the level of overall disciplines: $r_{\text{cited-citing}} = 0.90$, $r_{\text{cited-Price Index}} = -0.83$, $r_{\text{citing-Price Index}} = -0.89$). Literature with a long lifetime has high cited and citing half-life values but low Price Index, and vice versa, giving negative correlations between cited/citing half-life and Price Index.

Looking at the subdisciplines within computing, one finds the citation lifespan of the literature is shortest within Information Systems and longest within Theory & Methods (see Table 2). The variations among the subdisciplines of the various disciplines are generally small; on average, the cited half-life $\text{stddev} = 0.8$. The variation among the journals within a discipline is much greater; on average, the cited half-life $\text{stddev} = 2.2$. For computing journals, $\text{stddev} = 2.0$, with the extremes varying from cited half-life = 1.7 and citing half-life = 3.7 to >10 for both half-life measures.

Based on the assumption that everything is changing quickly throughout society, it is easy to believe that the scientific literature is becoming obsolete more quickly than it used to. However, a comprehensive study shows that the median citation age (citing half-life) of scientific publications has increased steadily since the 1970s.¹⁰ One likely reason for this increase is the availability of online bibliographic databases and the Internet, making it easier to access older references. A 2008 study reported, “The Internet appears to have lengthened the average life of academic citations by six to eight months.”¹¹ Another reason may be the significant increase in the number of references per article.¹⁰ Having space for more references allows for increasing the time period for included references.

The reported study¹⁰ focused on medical fields, natural sciences, and engineering. To study the evolution of the aging distribution of the computing literature compared to all other disciplines, I investigated cited half-lives from 2003 to 2007 (see the sidebar “How the Study Was Done”); JCR did not provide such information earlier than 2003. I found the cited half-

life of computing literature increased from 7.1 years in 2003 to 7.4 years in 2007 (4.7%), the fifth highest increase among the 22 disciplines. Geosciences was tops, with an increase from 8.2 years to 8.8 years (7.7%). The disciplines with the most decreasing cited half-life were Environment/Ecology and Engineering, with declines of 2.4% and 1.8%, respectively. The average increase among all disciplines was 0.1 year (1.9%). Hence, there seems to be a

trend that the age of useful computing literature is increasing, not decreasing relative to other disciplines.

The increasing interest in research related to environment and ecology may have contributed to less old work being cited in more recent issues of the related journals. Moreover, if my study is replicated in, say, five years, we may observe different trends; for example, the financial crisis at the time of this writing (2009) may contribute to more

Table 1. Half-lives and Price Index for all scientific disciplines.

Discipline	Mean Half-Life		Cited Half-Life		Citing Half-Life		Price Index	
	Years	Rank	Years	Rank	Years	Rank	%	Rank
Immunology	5.9	1	5.8	1	6.1	1	41.9	1
Molecular Biology and Genetics	6.5	2	6.2	2	6.7	2	29.3	13
Space sciences	6.6	3	6.3	3	6.8	3	40.1	2
Pharmacology	6.6	3	6.3	3	6.9	4	36.2	4
Biology and Biochemistry	6.7	5	6.5	7	6.9	4	35.3	6
Microbiology	6.8	6	6.3	3	7.2	7	35.3	5
Clinical Medicine	6.8	6	6.7	8	7.0	6	36.5	3
Chemistry	7.0	8	6.4	6	7.6	9	33.7	7
Neuroscience & Behavior	7.3	9	6.9	11	7.6	9	33.3	9
Physics	7.3	9	6.8	9	7.7	11	33.6	8
Multidisciplinary	7.3	9	6.8	9	7.8	12	33.1	10
Computer Science	7.5	12	7.4	14	7.5	8	31.7	11
Engineering	7.7	13	7.2	13	8.3	13	29.7	12
Environment/Ecology	7.9	14	7.4	14	8.4	15	26.5	17
Materials Science	7.9	14	7.1	12	8.8	17	28.7	14
Agricultural Sciences	8.1	16	7.5	16	8.7	16	25.5	19
Social Sciences, General	8.2	17	8.1	17	8.3	13	27.9	15
Plant & Animal Sciences	8.8	18	8.4	18	9.2	18	26.4	18
Psychiatry/Psychology	9.1	19	9.0	20	9.2	18	25.0	20
Geosciences	9.2	20	8.8	19	9.5	21	27.1	16
Economics and Business	9.6	21	9.9	22	9.2	18	24.8	21
Mathematics	9.7	22	9.5	21	9.8	22	23.9	22
Mean	7.7		7.3		8.0		31.2	

Table 2. Half-lives and Price Index for computing.

Subdiscipline	Mean Half-Life		Cited Half-Life		Citing Half-Life		Price Index	
	Years	Rank	Years	Rank	Years	Rank	%	Rank
Information Systems	6.8	1	6.7	2	6.8	1	35.1	1
Interdisciplinary Applications	7.0	2	6.2	1	7.8	4	31.1	4
Artificial Intelligence	7.6	3	7.2	3	8.0	5	28.2	6
Software Engineering	7.7	4	8.1	5	7.2	3	33.8	3
Hardware & Architecture	7.8	5	8.8	7	6.8	2	35.1	1
Cybernetics	7.9	6	7.2	3	8.5	7	26.1	7
Theory & Methods	8.3	7	8.6	6	8.0	5	29.7	5

research being done in economics and business, with more recent work being cited, or shorter half-lives.

Journals vs. conferences. I found an average of 5.9 years for the citing half-life of the 307 conference and workshop proceedings available in the ACM Digital Library. Their citing half-lives are shorter than for computing journals (7.5 years). The two main explanations for why conferences have shorter half-lives are shorter publication delay and fewer references per article.

Publication delay means the cited references grow older due to the publication process per se; that is, the references were younger when the article was submitted than when the article was published. A list of publication delays for computing journals, conferences, and other venues shows a clear tendency for journals to have longer delays than conferences (<http://cite-seer.ist.psu.edu/pubdelay.html>). The average publication delay of journals common to the CiteSeer list and JCR was 20 months. The average publication delay of the conferences common to the CiteSeer list and the ACM Digital Library was eight months. About one-third of the JCR journals and one-quarter of the ACM Digital Library conferences were included. It is unlikely these samples were biased with respect to publication delay. Hence, we can infer that the average difference in publication delay between computing journals and conferences is approximately one year, even though the increasing use of Web-based support tools in the review process of many journals may have contributed to slightly shorter publication delays today than when the list was assembled in 2003.

The 11,719 articles in the ACM conferences (as of 2008) include, on average, 16.1 references, while the 36,004 articles in the JCR computing journals include, on average, 27.1 (26.2 if review articles are excluded); that is, journals include 70% more references than conferences. Journal articles are also generally longer than conference articles; thus, more space is available for related work. Consequently, the citing half-lives of journals may be higher than the citing half-lives of conference proceedings due in part to journals citing more references.



One should take care criticizing or ignoring literature just because it is “old”; other criteria must be used to judge quality and relevance.



When calculating the half-lives of the conference proceedings, I excluded references to URLs because their year of “publication” was rarely indicated in their citations; moreover, for those URL references with the year indicated, it’s likely that the content of the actual Web site has changed, meaning we cannot necessarily use the indicated year to calculate the age of the content of a given Web site (unlike printed publications). However, another study¹⁶ investigated how long URLs are accessible by inspecting the URLs referenced in articles in *IEEE Computer and Communications* from 1995 to 1999, reporting, “A noteworthy parallel can be observed between the four years we calculated as the half-life of referenced URLs and five years given as the median citation age for computer science.” One may reasonably question the extent to which one is able to compare the accessibility of URLs with the inclusion of references in articles.

Nevertheless, the claim that the half-life in CS is five years is from four issues of the *Proceedings of the International Conference on Information Systems*.³ Due to difference between journals and conferences, it would be more correct to compare the four-year half-life of URLs with the citing half-life of 7.5 years in Table 1, as both figures result from analyzing journals. In this case, referenced articles would have a useful life approximately twice as long as the URLs. However, given that I found large variations in the citing half-lives between journals and conferences with respect to printed publications, one may find large variations in the half-lives of referenced URLs as well. Therefore, one should analyze much larger samples than only two journals to make a general statement.

Conclusion

My investigation found that the aging of the computing literature is not atypical compared with other scientific research disciplines, indicating that the research front in computing does not move more quickly than its counterpart in other disciplines. It is also a sign that computing is an established research discipline with long-lasting challenges and complex research problems taking years to solve. For example, developing software systems that

are reliable, efficient, user-friendly, and maintainable has been, and probably always will be, a grand challenge in computing research. Moreover, it typically takes 10 to 20 years for a technology to mature from being a good research idea to being widely used in practice.^{14,15} This fundamental aspect of computing, combined with the importance of software in modern society, means there is no reason funding for computing research should not be at a level comparable to that found in other scientific disciplines, including physics and clinical medicine.

These results have further consequences. First, half of the citations in the computing literature are more than seven years old. Publications older than seven years may be viewed as old but still considered relevant by the authors citing them. Therefore, one should take care criticizing or ignoring literature just because it is “old”; other criteria must be used to judge quality and relevance.

The relatively long cited half-life of computing literature also indicates that the time lag between submitting a paper to a journal and it being published in

that journal should not be a major concern; such work is rarely obsolete before publication. In any case, the delay may be significantly shorter in the future, as an increasing number of journals publish their articles online shortly after accepting them for publication.

My results also indicate that computing journals are not more likely to have their subscriptions cancelled or stored for a shorter time than journals of other scientific disciplines. There are significant variations, so decisions regarding particular journals must be based on more detailed information about the journals.

Here, I’ve discussed obsolescence at a coarse level (disciplines and sub-disciplines). It would be interesting to study obsolescence within categories of computing topics and research. For example, how does obsolescence vary between research that aims to solve (minor) practical problems and research that aims to develop comprehensive theories? However, this would require substantial effort, given there is no database that easily provides relevant data similar to what JCR provided for the study I’ve reported here.

Acknowledgments

I thank Chris Wright for help clarifying basic concepts and stimulating comments; Gilles Brassard and anonymous referees for valuable comments; and Alexander Ottesen, Birgitte Refsland, and Bjørnar Snoksrud for help collecting the reported data. 

References

1. Barnett, G.A. and Fink, E.L. Impact of the Internet and scholar age distribution on academic citation age. *Journal of the American Society for Information Science and Technology* 59, 4 (Feb. 2008), 526–534.
2. Burrell, Q. Stochastic modelling of the first-citation distribution. *Scientometrics* 52, 1 (Sept. 2001), 3–12.
3. Cunningham, S.J. and Bockock, D. Obsolescence of computing literature. *Scientometrics* 34, 2 (Oct. 1995), 255–262.
4. De Solla Price, D.J. Citation measures of hard science, soft science, technology and nonsense. In *Communication Among Scientists and Engineers*, C.E. Nelson and D.K. Pollack, Eds. D.C. Heath and Company, Lexington, MA, 1970, 3–22.
5. De Solla Price, D.J. Networks of scientific papers: The pattern of bibliographic references indicates the nature of the scientific research front. *Science* 149, 3683 (July 1965), 510–515.
6. Glänzel, W. Towards a model for diachronous and synchronous citation analyses. *Scientometrics* 60, 3 (Dec. 2004), 511–522.
7. Goodrum, A.A., McCain, K.W., Lawrence, S., and Giles, C.L. Scholarly publishing in the Internet age: A citation analysis of computer science literature. *Information Processing and Management* 37, 5 (Sept. 2001), 661–675.
8. ISI Web of Knowledge. *Journal Citation Reports on the Web 4.2*. The Thomson Corporation, 2008; <http://www.isiknowledge.com/JCR>
9. Ladwig, J.P. and Sommes, A.J. Using cited half-life to adjust download statistics. *College & Research Libraries* 66, 6 (Nov. 2005), 527–542.
10. Larivière, V., Archambault, É., and Gingras, Y. Long-term variations in the aging of scientific literature: From exponential growth to steady-state science (1900–2004). *Journal of the American Society for Information Science and Technology* 59, 2 (Jan. 2008), 288–296.
11. Lazowska, E.D. and Patterson, D.A. An endless frontier postponed. *Science* 308, 5723 (May 2005), 757.
12. Misa, Y.J. Understanding ‘how computing has changed the world’. *IEEE Annals of the History of Computing* 29, 4 (Oct.–Dec. 2007), 52–63.
13. Moed, H.F. *Citation Analysis in Research Evaluation*. Springer, Dordrecht, The Netherlands, 2005.
14. Osterweil, L.J., Ghezzi, C., Kramer, J., and Wolf, A.L. Determining the impact of software engineering research on practice. *IEEE Computer* 41, 3 (Mar. 2008), 39–49.
15. Redwine Jr., S.T. and Riddle, W.E. Software technology maturation. In *Proceedings of the Eighth International Conference on Software Engineering* (London, Aug. 28–30). IEEE Computer Society Press, Los Alamitos, CA, 1985, 189–200.
16. Spinellis, D. The decay and failures of Web references. *Commun. ACM* 46, 1 (Jan. 2003), 71–77.
17. Stinson, R. and Lancaster, F.W. Synchronous versus diachronous methods in the measurement of obsolescence by citation studies. *Journal of Information Science* 13, 2 (Apr. 1987), 65–74.
18. Száva-Kovács, E. Unfounded attribution of the ‘half-life’ index-number of literature obsolescence to Burton and Keble: A literature science study. *Journal of the American Society for Information Science and Technology* 53, 13 (Nov. 2002), 1098–1105.
19. Weingarten, F. Government funding and computing research priorities. *ACM Computing Surveys* 27, 1 (Mar. 1995), 49–54.

Dag I.K. Sjøberg (dagsj@ifi.uio.no) is a professor of software engineering in the Department of Informatics at the University of Oslo, Norway.

© 2010 ACM 0001-0782/10/0900 \$10.00

How the Study Was Done

I used Thomson’s *JCR Science Edition* (6,417 journals in 172 categories) and *Social Sciences Edition* (1,865 journals in 55 categories) for 2007, with most journals in the (natural) sciences covered in the selection. The coverage in the social sciences was less comprehensive.¹³ To comprehensively compare the overall computing discipline with other scientific disciplines, I first aggregated the JCR journal categories into 22 disciplines, per *ScienceWatch* (<http://sciencewatch.com/about/met/fielddef/>) and discarded eight of the 227 JCR categories because I could not fit them into the scheme of the aggregated disciplines.

JCR provided citation data at the level of both journals and categories but did not provide half-lives for new journals or journals cited fewer than 100 times. Among the 382 CS journals, 8% and 2%, respectively, lacked cited and citing half-lives. In the calculations of the aggregated results by discipline (see Table 1), I weighted the categories with respect to their number of journals.

For half-lives >10, JCR used only the value 10 in the calculation of aggregated half-lives. In the aggregation of half-life values from the JCR categories into the disciplines, I used the same approximation. A half-life “>10” was reported for individual categories in nine of the 22 disciplines; on average, 25% of the categories had the value “>10.” Note that even if these nine disciplines were registered with exact values, it would not affect the CS position relative to the other disciplines in Table 1.

JCR focused on journals for citation data. However, though a study⁷ reported that conference proceedings were less cited in the computing literature than books and journals, conferences play an important role in computing research. I therefore investigated the proceedings in the ACM Digital Library (from conferences and workshops in 2007) to make the data comparable with the data from the JCR 2007 edition. I included all scientific papers with at least one reference where the publication year was given; I thus excluded 2.7% of the papers on this ground. A script crawled the Web sites and extracted the references of each article in 307 proceedings. I then analyzed the output using a regular expression to identify the year of publication, enabling me to calculate the citing half-life. The 0.9% of the references lacking a clear year of publication required manual inspection.

DOI:10.1145/1810891.1810910

The same component isolation that made it effective for large distributed telecom systems makes it effective for multicore CPUs and networked applications.

BY JOE ARMSTRONG

Erlang

ERLANG IS A concurrent programming language designed for programming fault-tolerant distributed systems at Ericsson and has been (since 2000) freely available subject to an open-source license. More recently, we've seen renewed interest in Erlang, as the Erlang way of programming maps naturally to multicore computers. In it the notion of a process is fundamental, with processes created and managed by the Erlang runtime system, not by the underlying operating system. The individual processes, which are programmed in a simple dynamically typed functional programming language, do not share memory and exchange data through message passing, simplifying the programming of multicore computers.

Erlang² is used for programming fault-tolerant, distributed, real-time applications. What differentiates it from most other languages is that it's a concurrent programming language; concurrency belongs to the language, not to the operating system. Its programs are collections of parallel processes cooperating to solve a particular problem that can be created quickly and have only limited memory

overhead; programmers can create large numbers of Erlang processes yet ignore any preconceived ideas they might have about limiting the number of processes in their solutions.

All Erlang processes are isolated from one another and in principle are "thread safe." When Erlang applications are deployed on multicore computers, the individual Erlang processes are spread over the cores, and programmers do not have to worry about the details. The isolated processes share no data, and polymorphic messages can be sent between processes. In supporting strong isolation between processes and polymorphism, Erlang could be viewed as extremely object-oriented though without the usual mechanisms associated with traditional OO languages.

Erlang has no mutexes, and processes cannot share memory.^a Even within a process, data is immutable. The sequential Erlang subset that executes within an individual process is a dynamically typed functional programming language with immutable state.^b Moreover, instead of classes, methods, and inheritance, Erlang has modules that contain functions, as well as higher-order functions. It also includes processes, sophisticated error handling, code-replacement mechanisms, and a large set of libraries.

Here, I outline the key design criteria behind the language, showing how they are reflected in the language itself, as well as in programming language technology used since 1985.

Shared Nothing

The Erlang story began in mid-1985 when I was a new employee at the Ericsson Computer Science Lab in Stock-

a The shared memory is hidden from the programmer. Practically all application programmers never use primitives that manipulate shared memory; the primitives are intended for writing special system processes and not normally exposed to the programmer.

b This is not strictly true; processes can mutate local data, though such mutation is discouraged and rarely necessary.



holm charged with “doing something about how we write software.” Ericsson had a long tradition of building highly reliable fault-tolerant systems (telephone exchanges) specified to have at most four minutes of downtime per year and system software that could be upgraded without stopping the system.

How would we do it? The question was answered in the mid-1970s and has been the same ever since. The system would have to be constructed from physically isolated components communicating through well-defined “pure” protocols. The word “pure” has special significance, meaning that after a message passes there should be no dangling pointers or data references to data structure residing on other machines.

Fault-tolerance is achieved like this: If a machine crashes, the failure is detected by another machine in the network. The machine (or machines) detecting the failure must have sufficient data to take over from the machine that crashed and continue with the application. Users should not notice the failure.

This technique was used by Jim Gray¹⁰ in the design of the fault-tolerant Tandem computer. The Tandem hardware architecture was similar to the software architecture used to build Erlang applications. Using failure detection plus replication to make reliable systems has a long history.¹¹

Now assume we have a single machine, and the probability that it will fail during some time period is 10^{-3} . If we have two identical isolated machines, then the probability they both will fail in the same time period is 10^{-6} , with three machines 10^{-9} , and so on. Component isolation is the key to building reliable systems. Individual components might fail, but the probability that all components will fail at the same time can be made arbitrarily small by having a sufficiently large number of replicated components.

This approach works for hardware, but what about for software? If 10 copies of some software run on 10 different isolated machines, won't they all fail for the same reason if they all have the same software and are trying to solve the same problem? Of course they will, but in the systems we build, this is not

a problem. Imagine a system in which 10,000 transactions are in progress simultaneously, including telephone calls, Web sessions, database queries, anything. Each transaction could be running the same software, but each instance of the software will also have some private state. An individual process crashing due to a software error is not problematic, provided all other processes in the system (where no errors have occurred) are not affected by the crash.

Building fault-tolerant software requires the same trick used to build fault-tolerant hardware. We arrange for one process to observe the behavior

» key insights

- **Message-passing systems scale easily, are surprisingly efficient, and can be made fault tolerant through replication over several isolated machines.**
- **Non-defensive programming and Erlang's “let it crash” style of programming lead to clear, compact code.**
- **Upgrading systems without taking them out of service has been practiced in the telecom world for years; Erlang makes it relatively easy.**

of another process. The observing process must be able to detect failures in the observed process and take over in the event of an error.

We also forbid dangling pointers and shared data structures between processes. The entire system is constructed so the observed processes and their observers need not even be on the same machine. For example, in distributed Erlang, processes can be scattered over physically separated nodes and behave semantically as if they were on the same node. The only difference is in the pragmatics of the system; the latency of an operation performed on a local process and a process located on some physically separated node are very different.

This property of Erlang processes means programs can be developed on a single node and deployed on a cluster without major changes to most of the software in the system.

In light of such considerations, we concluded (in 1986) that in order to program fault-tolerant applications Erlang would need four key properties^{4,9}:

- ▶ Isolated processes;
- ▶ Pure message passing between processes;
- ▶ The ability to detect errors in remote processes; and
- ▶ A method for determining what error caused a process to crash.

We did not want to use shared memories, mutexes, or semaphores, so our only method of process synchronization was through message passing—viewed by most programmers at the time as a crazy method for designing systems. The principal objection was efficiency; copying messages between processes (instead of using shared memory) was considered horrendously inefficient. The counterargument was that shared memory was something preventing fault-tolerance. I have always believed that systems should be made to work correctly before they are made fast. Fault-tolerance in the presence of both hardware and software errors must be addressed ahead of efficiency.

Fast-forward almost 25 years from 1986 to see that networked applications are extremely common and multicore computers are everywhere. As the number of cores increases so does the need for isolation throughout the

system. Small isolated computations are easily allocated to a pool of cores. Shared memory translates to cache-misses in multicore computers. If a process running on one core of a multicore computer wants to access data in the cache of a physically distant core, pipeline stalls will occur, and the entire operation will take much longer than if the memory had been available locally.

Erlang today is well-placed for programming multicore CPUs. Faced with a multicore CPU, most programmers turn legacy code into a parallel program. Erlang programmers face an entirely different problem. They already have a parallel program, but it might have some sequential bottlenecks, so their job is to find the bottlenecks.

Here, I explore the language, along with some of the more interesting applications that have been written in it. Though Erlang started in the telecom world, it has escaped to wider pastures, rather like Unix and C.

Erlang View of the World

The Erlang view of the world is that everything is a process that lacks shared memory and influences one another only by exchanging asynchronous messages. This view is broadly similar to the actors model proposed by Gul Agha.¹ Each process has a mailbox to which messages can be sent. Messages are retrieved from the mailbox with a receive statement or pattern-matching construction that removes messages matching a particular pattern in the mailbox and can also be used to selectively remove messages from the mailbox.

Hardware in Erlang is interfaced through processes. A process that controls hardware has two interfaces: one toward the Erlang system, where it behaves as a regular Erlang process, the other toward the hardware controlled through a port providing an I/O channel to the outside world. All communication with the outside takes place through ports.

Foreign-language software (not in Erlang) cannot be linked to the Erlang kernel but must be run in a separate operating system process that executes outside the Erlang runtime system and is interfaced through a port. Security is the reason for not linking foreign-language code into the kernel; we do not

want errors in external code crashing the Erlang runtime system.

Erlang View of Errors

Erlang differs from most other programming languages in the way it handles errors. An Erlang system typically consists of large numbers of lightweight processes. It is of no particular consequence if any one of them dies. The recommended way of programming is to let failing processes crash and other processes detect the crashes and fix them.

Erlang has a safe type system. Data structures are dynamically typed, and it is impossible to create corrupt data structures. Extensive user checking of data structures is unnecessary, since the worst that can happen is an individual process might crash if it performs an illegal operation. The important thing to note is that the crash of one process does not affect any other unlinked process in the system.

However, being type-safe does not solve all programmer problems; for example, exception handlers must still be written to correct type errors, and sets of observing processes must be created to correct errors caused when processes crash due to type errors. Some of these errors could have been caught by static type checking, but adding complete static type checking to Erlang would change the flavor of the language and make upgrading dynamic code and other things virtually impossible.

In a single-threaded application one has only one chance to correct an error, so the consequence of not correcting an error is that an entire application might fail, thus single-threaded applications and languages take great care to fix errors locally. With thousands of processes at one's disposal one is less concerned about the failure of individual processes than about detection and correction of errors. The system is divided into worker processes that perform computations and supervisor processes that check that the worker processes are behaving correctly.

Erlang has an internal mechanism, or “link,” that provides a form of inter-process error detection and performs as an error-propagation channel. If process A is linked to process B and

process A dies, then an error signal will be sent to process B, and vice versa.

The ability to monitor a process provides a clue for building reliable systems. The idea is to try to solve the problem, but if the processes in the solution cannot do the job, the system tries to solve a simpler problem. “Cannot do the job” means detecting the failure of a process; the system detects such failures and tries to solve a simpler problem.

One layer of the system usually performs the application logic, and an error-trapping layer monitors the application and restores it to a safe state if an error occurs. This application structure is formalized in the Erlang Open Telecom Platform (OTP) system using so-called supervision trees providing a precise description of what is to happen if a computation fails. OTP applications organize problems into tree-structured groups of processes, letting the higher nodes in the tree monitor and correct errors occurring in the tree’s lower nodes.

Erlang Programs

Erlang was first implemented in Prolog⁶ in 1986, and thus many of the syntactic conventions used in Erlang come from Prolog; Erlang’s syntactic conventions include:

Variables. When variables, or single-assignments (written starting with an uppercase letter like `Day` and `File`), acquire a value, that value cannot be changed; variables acquire values in successful pattern-matching operations;

Atoms. Used to represent constants, they are similar to enumerated types in Java and C and written starting with a lower-case letter; for example, `monday`, `orange`, and `cat` are atoms;

Tuples. Like structs in C and used for storing fixed numbers of items, tuples are written in curly brackets; for example, `{Var, monday, 12}` is a tuple containing a variable atom and an integer; and

Lists. Used for storing variable numbers of items, lists are written enclosed in square brackets; for example, `[a,X,b,Y]` is a list containing two atoms and two variables.

Erlang’s syntax is designed to make it easy to express parallel computations. Here, I jump in the deep end



The recommended way of programming is to let failing processes crash and other processes detect the crashes and fix them.



of Erlang program development with a code fragment that creates a counter process. Many of the examples are from my 2007 book *Programming in Erlang*² and contain all the gruesome details one would need to write Erlang code. I begin by creating a counter process:

```
Pid = spawn(fun() -> counter(0)
end),
```

`spawn(Fun)` means “create a parallel process that evaluates `Fun`,” or an Erlang function.

The function `counter(N)` in Figure 1 waits for one of two messages: If the process is sent the message `tick`, it calls `counter(N+1)`. If it is sent the message `{From, read}` it replies by sending a message `{self(), N}` to the process `From` and then calls `counter(N)`. The notation `A ! B` means send the message `B` to the process `A`, `self()` is the process identity of the process running the counter function and

```
receive
  Pattern1 ->
    Actions1;
  Pattern2 ->
    Actions2;
  ...
end
```

means wait for a message. If the next message matches `Pattern1`, then execute the code `Actions1`; otherwise if the message matches `Pattern2`, then execute the code `Actions2`, and so on. If no pattern is matched, then queue the message for later and wait for the next message.

To bump the counter, some process that knows the name of the process executes the code:

```
Pid ! tick.
```

To read the counter, we evaluate:

```
Pid ! {self(), read},
receive
  {Pid, Result} ->
    Result
end
```

We send a `{self(), read}` message to the counter process, then wait for a

Figure 1. A simple counter process.

```

counter(N) ->
  receive
    tick ->
      counter(N+1);
  {From, read} ->
    From ! {self(), N},
    counter(N)
  end.

```

return message {Pid, Result}. Variables in Erlang are bound only once and thereafter can never be changed, so when one enters the receive statement, Pid has a value; it must have a known value, since otherwise Pid ! .. would be meaningless. We must know the identity of a process in order to send it a message.

The receive statement then waits for a message {Pid, Result} where Pid is a bound variable and Result is an unbound variable. This code fragment means “Wait for a message that is a tuple with two arguments where the first argument matches Pid and bind the value of the second argument in the tuple to the variable Result. That is, the code fragment waits for a message from the process Pid and queues any other messages that might arrive while waiting for the message.

This code fragment occurs so often it’s been given a name and made into a library function:

```

rpc(Pid, Request) ->
  Pid ! {self(), Request},
  receive
    {Pid, Response} ->
      Response
  end.

```

It is simply a remote procedure call.

Erlang has no built-in mechanism for doing remote procedure calls, but one can easily program a remote procedure call using the built-in primitives send and receive. Why are built-in primitives important? Because we can roll our own interprocess communication mechanisms. If we want to do two remote procedure calls in parallel, it could be done like this:

```

Pid1 ! {self(), Request1},
Pid2 ! {self(), Request2},
receive
  {Pid1, Response1} ->
    Response1
end,
receive
  {Pid2, Response2} ->
    Response2
end,
...

```

This code is non-blocking since receive automatically queues any out-of-order messages sent to the processes. If Pid1 replies first, then the first receive clause is triggered, and execution steps to the second receive statement and waits for the second process to reply. If Pid2 replies first, then the message is queued; once Pid1 replies, the first receive statement is satisfied and the program steps to the second receive statement, but the message will have been saved and queued, so the second receive statement is triggered immediately. The net result is that on completion of the code fragment both messages will have been received irrespective of the order in which they were sent. The time spent waiting is the longer of the response times from the two processes.

Note, too, if the program had exposed only a composite remote proce-

dures call function, such programming would be more difficult, since two intermediate processes would have been spawned, where each performed a remote procedure call, and the results would then have to be combined. With three or more processes, coordinating the actions of the parallel processes would be difficult to program, were it not for the queuing mechanism built into the Erlang receive statement.

Detecting errors. Recall that in order to build reliable systems one must be able to remotely detect errors.^c Figure 2 defines a function that can detect an error in a remote process and perform an action on detecting the error, and on `_exit(Pid, F)` creates a process that monitors the process Pid. If the monitored process dies with reason Why, the newly created process evaluates the function `F(Why)`, and `process_flag(trap_exit, true)` turns the current process into a “system process” that can trap exit signals. The statement `link(Pid)` sets up a “link” to the process Pid. A link is an error-propagation channel, and `link(Pid)` means “if the process Pid dies, send me an exit signal.” An exit signal is an out-of-band message sent when a process dies. Processes normally die when they receive out-of-band exit signals, but because the process evaluated `process_flag(trap_exits, true)`, it became a system process, and thereafter the exit signal can be received as a message containing a {‘EXIT’, Pid, Why} tuple.

The function `on_exit` is the workhorse needed to build fault-tolerant code. Using `on_exit` allows one to build a hierarchical tree of processes. Some processes do the work, and other processes monitor the processes that do the work and fix things up if the worker processes die.

Recall that the Erlang philosophy is “Let it crash”; in fact, processes that cannot perform the task they were told to do should crash immediately. Another process will correct the error. This is exactly the opposite of defensive programming but leads to a clean

^c Local error detection is no good; the local machine might have crashed and cannot perform error recovery, so the error must be detected on a remote machine unaffected by the crash.

Figure 2. A process monitor.

```

on_exit(Pid, F) ->
  spawn(fun() -> monitor(Pid, F) end).

monitor(Pid, F) ->
  process_flag(trap_exit, true),
  link(Pid),
  receive
    {‘EXIT’, Pid, Why} ->
      F(Why)
  end.

```

separation of interest between code that does the job and code that cleans up an error when it occurs. Erlang does not provide an `on_exit` function, but it is easy to program one using the Erlang's built-in primitives.

Dynamic code upgrade. One thing users and developers alike want to do is run their systems forever. Assuming things change, they will also want to change the code in a running system, but how? Imagine a simple server written as follows:

```
loop(State, F) ->
  receive
    {From, Request} ->
      {Response, State1}
      =F(Request, State),
      From ! {self(), Response},
      loop(State1, F)
  end.
```

This server is a simple extension to the counter process in Figure 1. The server process has state `State` and a processing function `F`. We could create a process that evaluates this loop like this:

```
F1 = fun(N, State) -> {N*N,
State+1} end,
Pid = spawn(fun() -> loop(0,
F1) end,
```

The processing function `F1` returns the square of its first argument and keeps a running total of the number of requests to the server.

The code in the server cannot be changed, but a small addition can be made to allow for dynamic code upgrade by adding a `{newFunction, F1}` pattern to the receive statement:

```
loop(State, F) ->
  receive
    {newFunction, F1} ->
      loop
        (State, F1);
    {From, Request} ->
      {Response, State1}
      = F(Request, State),
      From ! {self(), Response},
      loop(State1, F)
  end.
```

Now a new processing function can be sent to the server without interrupting it; for example, we could write:

```
F1 = fun(N, State) -> {N*N,
State+1} end,
Pid = spawn(fun() -> loop(0,
F1) end),
...
... some time later
...
F2 = fun(N, State) -> {N*N*N,
State+1} end,
Pid ! {newFunction, F2},
...
```

This new function dynamically upgrades the code in the server.

Adding transactions. Adding transactions is easy. In a transaction, either state is modified if it works or there is no change to the state if the transaction fails. To implement this, we add a try-catch-end block to the inner part of the receive statement:

```
loop(State, F) ->
  receive
    {newFunction, F1} ->
      loop(State, F1);
    {From, Request} ->
      try F(Request, State) of
        {Response, State1} ->
          From ! {self(), Response},
          loop(State1, F)
      catch
        _:Why ->
          exit(From, crash)
          loop(State, F)
      end
  end.
```

The evaluation of `F(Request, State)` is wrapped in a try-catch-end block. If the evaluated function raises an error, then the process evaluates the statement `exit(Pid, crash)`, which sends an exit signal to the process that caused the exception; thereafter, `loop(State, F)` is called, or recurs with the original value of the state.

The sequential part of Erlang is a functional language that does not allow the mutation of state. Because state cannot be mutated, an Erlang function can always revert to a previous state of the computation by accessing the original variable that referred to the state.

Finally, note the effect of tail-recursion. All server loops in the example code finish with tail calls. Once a tail call is made there is no going back; tail calls do not create additional stack

frames, since there is nowhere to return to, and a new stack frame is not required. Having made a tail call, all local variables in the current context can be garbage-collected, allowing tail-recursive loops to run indefinitely without consuming stack space.

Open Telecoms Platform

OTP is a large set of libraries written mostly in Erlang bundled together with the Erlang distribution. OTP can be viewed as an application middleware package that simplifies writing large Erlang applications. Recall I mentioned language primitives that could be used to build simple functions that encapsulate errors, showing how to build a simple function `on_exit` that could be used to evaluate a specific function if an error occurred in some other process.

Functions like `on_exit`, while useful and good to include in books on programming languages, are not the stuff from which large systems are built. If a software component in a large enterprise system fails, the error report must be kept forever and the system restarted. If a code upgrade fails, the entire system must be automatically rolled back to a previous state in a controlled manner.

Organizations employing large teams of programmers cannot let individual programmers invent their own error-handling mechanisms and ways of dynamically upgrading code. The OTP libraries are thus an attempt to formalize a large body of design knowledge into workable libraries that provide a standardized way of performing the most common tasks needed to build a reliable system.

OTP is the third total rewrite of a system of libraries in Erlang designed for building telecom systems.^{3,4} The 2010 OTP system includes 49 subsystems, each a powerful tool in its own right. Typical subsystems are `mnesia` (a real-time relational database), `megaco` (an H.248 stack), and `docbuilder` (a tool to make documentation), along with sophisticated analysis-test and analysis tools.

Because a large number of Erlang programs are written in a pure functional programming style, they are able to perform sophisticated analysis and transformations. For example, the

dialyzer¹⁴ is a type-checking program that performs static analysis of Erlang programs, finding type errors (if there are any) in them. The test tool Quick-Check⁸ generates random test cases from a specification of the formal properties of a program, and the tool Wrangler¹³ can be used to refactor Erlang programs.

Erlang Distribution

Ever since Erlang was first released into the public domain in 2000, it has been supported by an internal product-development group within Ericsson. Following the release of Open Source Erlang (<http://www.erlang.org/>), the language spread slowly for several years but has recently seen a dramatic upturn in the number of users and applications. This growth corresponds to a similar upturn in interest in Haskell (<http://www.haskell.org/>), a strictly typed lazy polymorphic programming language.

Two other languages in the same functional language school are OCaml and F#. The simultaneous increase in interest in different forms of functional programming can be seen as evidence that functional programming has come of age and is transitioning from the academic world to industrial practice. Industrial projects and the formation of new companies using Erlang as core technology reflect the more interesting developments. Erlang can be downloaded from <http://www.erlang.org/>, including the OTP system and a large number of tools.

Experience

In OO languages, objects are used to structure applications. In Erlang applications, processes are used for structuring, a technique I call “concurrency oriented programming,” or COP.⁵ The idea of building systems from communicating components is not new. Tony Hoare’s Communicating Sequential Processes¹² described how sets of concurrent processes could be used to model applications, and programming languages like Occam¹⁵ that were based on it explored the idea. Erlang is conceptually similar to Occam, though it recasts the ideas of CSP in a functional framework and uses asynchronous message passing



Not surprising, the leading uses of Erlang outside telecom all involve communications and reliable data storage.



instead of the synchronous message passing in CSP.

Processes in COP systems are isolated, responding only to messages and resulting in systems that are easy to understand, program, and maintain. Several fairly large systems written in Erlang enforce this idea. Several major product developments are based on Erlang, the largest being the AXD301 an asynchronous transfer mode (ATM) switch developed by Ericsson. Outside Ericsson, Erlang is being used by a large number of start-ups and is the principle technology of several new companies in Stockholm.

AXD301. The AXD301 switch⁹ has scalable capacity ranging from 10Gbit/sec to 160Gbit/sec and modular architecture and was written in distributed Erlang. Built by a large programming team, it has more than 1.6 million lines of Erlang code, showing that COP as a structuring method and Erlang as a programming language scale to large systems. One reason it scales so well is the architecture. At one level of abstraction, it can be viewed as a system of components that communicate through pure message passing. The lack of shared state and division of the system into well-isolated communicating components make it easy to understand the system’s overall architecture and isolate problems within the system.

When a message is sent into a component, we expect a certain response, or message, from it. If this does not happen, the error lies within the component. Opening it could reveal the same internal structure found on the outside, just a set of communicating components. “Opening a component” can be performed repeatedly until a misbehaving Erlang process is found. There is no magic. Making reliable systems from isolated components leads to systems that are easy to understand and manageable in both small- and large-scale projects.

Instant messaging. One problematic area in Internet applications where Erlang has found notable success is implementing instant-messaging systems. An IM system looks at first approximation very much like a telephone exchange. IM and telephone exchanges must both handle very large numbers of simultaneous transac-

tions, each involving communication with a number of simultaneously open channels. The work involved in parsing and processing the data on any one channel is small, but handling many thousands of simultaneous channels is a technical challenge.

Erlang's usefulness in IM is demonstrated by three projects:

MochiWeb (<http://code.google.com/p/mochiweb>). Designed for building lightweight HTTP servers developed by MochiMedia for high-throughput, low-latency analytics, and ad servers, this Erlang library helps power Facebook chat among more than 70 million users;

Ejabberd (<http://www.ejabberd.im>). Written by Alexey Shchepin, this Erlang implementation of the XMPP protocol is the most widely used open source XMPP server; and

RabbitMQ (<http://www.rabbitmq.com>). This Erlang implementation of the Advanced Message Queuing Protocol standard provides reliable asynchronous message passing at Internet scale.

Schema-free databases. In traditional databases, data is stored in rectangular tables, where the items in a table are instances of simple types (such as integers and strings). Such storage is not particularly convenient for storing an associative array or arbitrary tree-like structure. Examples of the former are JavaScript JSON data structures (called hashes in Perl and Ruby and maps in C++ and Java) and of the latter XML parse trees. These objects are difficult to store in a regular tabular structure. Erlang has for a long time had its own database, called *mnesia*, that includes table storage but allows any item in a table cell to also be an arbitrary Erlang data structure.

Databases implemented in Erlang are particularly well-suited for such storage, especially when they interface with some form of communicating agent. Three notable databases are implemented in Erlang:

CouchDB (<http://incubator.apache.org/couchdb/>). Written by Damien Katz, "Apache CouchDB is a distributed, fault-tolerant, schema-free document-oriented database accessible via a RESTful HTTP/JSON API." It provides robust, incremental replication with bidirectional conflict detection and resolution, queryable and index-

able through a table-oriented view engine, with JavaScript acting as the default view-definition language;

Amazon SimpleDB (<http://aws.amazon.com/simpledb/>). This Web service runs queries on structured data in real time; and *Scalaris*.¹⁶ This scalable, transactional, distributed key-value store has a peer-to-peer architecture for supporting reliable transactions with ACID properties.

CouchDB and Scalaris are open source projects; SimpleDB is a closed-source commercial service.

Sweet spot. Taking in the six projects described here reveals a pattern of communication with complex data structures being passed over the network. The number of clients wanting simultaneous access to the system is potentially huge, with hundreds of thousands to millions of users. The data stores must therefore be reliable and the data protocols extensible. Not surprising, this is the Erlang "sweet spot" for supporting system development. Erlang was developed for building high-performance telecom switches, with hundreds of thousands of users accessing the system simultaneously. Data structures are complex, and the system must be able to store data in a reliable manner, recovering from local failures and scaling clusters to manage varying demand. Erlang was designed to do all these things, with the intended applications domain of carrier-class telecoms systems. Also not surprising, the leading uses of Erlang outside telecom all involve communications and reliable data storage. In an abstract sense, what these projects do is serialize data terms into a transportable format (marshalling and unmarshalling), transport the data over the network, and store the data in some kind of persistent storage medium.

Beyond the sweet spot, several applications that have nothing to do with fault tolerance have also gained popularity; for example, *Wings* (<http://www.wings3d.com>), a 3D graphics modeling program written by Björn Gustavsson, and *Nitrogen* (<http://nitrogenproject.com/>), a Web-development framework written by Rusty Klophaus, show that Erlang is useful as a general-purpose programming language.

Acknowledgments

I thank Ericsson Telecom for its contribution to the development of Erlang over the years. 

References

1. Agha, G. *Actors: A model of concurrent computation in distributed systems*. In *MIT Series in Artificial Intelligence*. MIT Press, Cambridge, MA, 1986.
2. Armstrong, J. *Programming Erlang: Software for a Concurrent World*. The Pragmatic Bookshelf, Raleigh, NC, 2007.
3. Armstrong, J. A history of Erlang. In *Proceedings of the Third ACM SIGPLAN Conference on the History of Programming Languages* (San Diego, CA, June 9–10). ACM Press, New York, 2007.
4. Armstrong, J. *Making Reliable Distributed Systems in the Presence of Errors*. Ph.D. Thesis, Royal Institute of Technology, Stockholm, 2003.
5. Armstrong, J. Concurrency-oriented programming in Erlang. Invited Talk at the Lightweight Languages Workshop (Cambridge MA, Nov. 9, 2002).
6. Armstrong, J.L., Virding, S.R., and Williams, M.C. Use of Prolog for developing a new programming language. In *Proceedings of the First Conference on the Practical Application of Prolog* (London, Apr. 1–3). Association for Logic Programming, 1992.
7. Blau, S. and Rooth, J. Axd 301: A new-generation ATM switching system. *Ericsson Review* 1 (1998).
8. Claessen, K. and Hughes, J. Quickcheck: A lightweight tool for random testing of Haskell programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming*. ACM Press, New York, 2000, 268–279.
9. Däcker, B. *Concurrent Functional Programming for Telecommunications: A Case Study of Technology Introduction*. Licentiate Thesis. Royal Institute of Technology, Stockholm, 2000.
10. Gray, J. *Why Do Computers Stop and What Can Be Done About It?* Tech. Rep. 85.7. Tandem Computers, Inc., 1985.
11. Guerraoui, R. and Schiper, A. Fault tolerance by replication in distributed systems. In *Proceedings of the Conference on Reliable Software Technologies*. Springer Verlag, 1996, 38–57.
12. Hoare, C.A.R. *Communicating Sequential Processes*. Prentice Hall, Upper Saddle River, NJ, 1985.
13. Li, H., Thompson, S., Orosz, G., and Toth, M. Refactoring with Wrangler: Data and process refactorings and integration with Eclipse. In *Proceedings of the Seventh ACM SIGPLAN Erlang Workshop* (Victoria, BC, Sept. 27). ACM Press, New York, 2008, 61–72.
14. Lindahl, T. and Sagonas, K. Detecting software defects in telecom applications through lightweight static analysis: A war story. In *Proceedings of the Second Asian Symposium* (Taipei, Taiwan, Nov. 4–6). Springer, 2004, 91–106.
15. *Occam Programming Manual*. Prentice Hall, Upper Saddle River, NJ, 1984.
16. Schütt, T., Schintke, F., and Reinefeld, A. Scalaris: Reliable transactional p2p key/value store. In *Proceedings of the Seventh ACM SIGPLAN Workshop on Erlang* (Victoria, BC, Sept. 27). ACM Press, New York, 2008, 41–48.
17. Wiger, U., Ask, G., and Boortz, K. World-class product certification using Erlang. In *Proceedings of the 2002 ACM SIGPLAN Workshop on Erlang* (Pittsburgh, PA). ACM Press, New York, 2002, 24–33.
18. Wiger, U. Fourfold increase in productivity and quality: Industrial-strength functional programming in telecom-class products. In *Proceedings of the Workshop on Formal Design of Safety Critical Embedded Systems* (Münich, Mar. 21–23, 2001).

Joe Armstrong (joe.armstrong@ericsson.com) is an expert in software architectures and programming languages in Business Unit Networks at Ericsson, Stockholm, Sweden.

A call for the perfect marriage between classical performance evaluation and state-of-the-art verification techniques.

BY CHRISTEL BAIER, BOUDEWIJN R. HAVERKORT,
HOLGER HERMANN, AND JOOST-PIETER KATOEN

Performance Evaluation and Model Checking Join Forces

CONSIDER A MAJOR news Web site like BBC or CNN. Typically, such a site is equipped with a number of machines serving as front-ends to receive incoming requests together with some application servers such as database engines to handle these requests. When a new request arrives, to which server does the dispatcher have to route it? To the machine with the shortest queue; that is, the queue with the minimal number of outstanding requests? This might be the best decision most times, but not in cases where some of the requests in the shortest queue happen to require a very long service time, for example, because they involve very detailed queries. And what to do when the

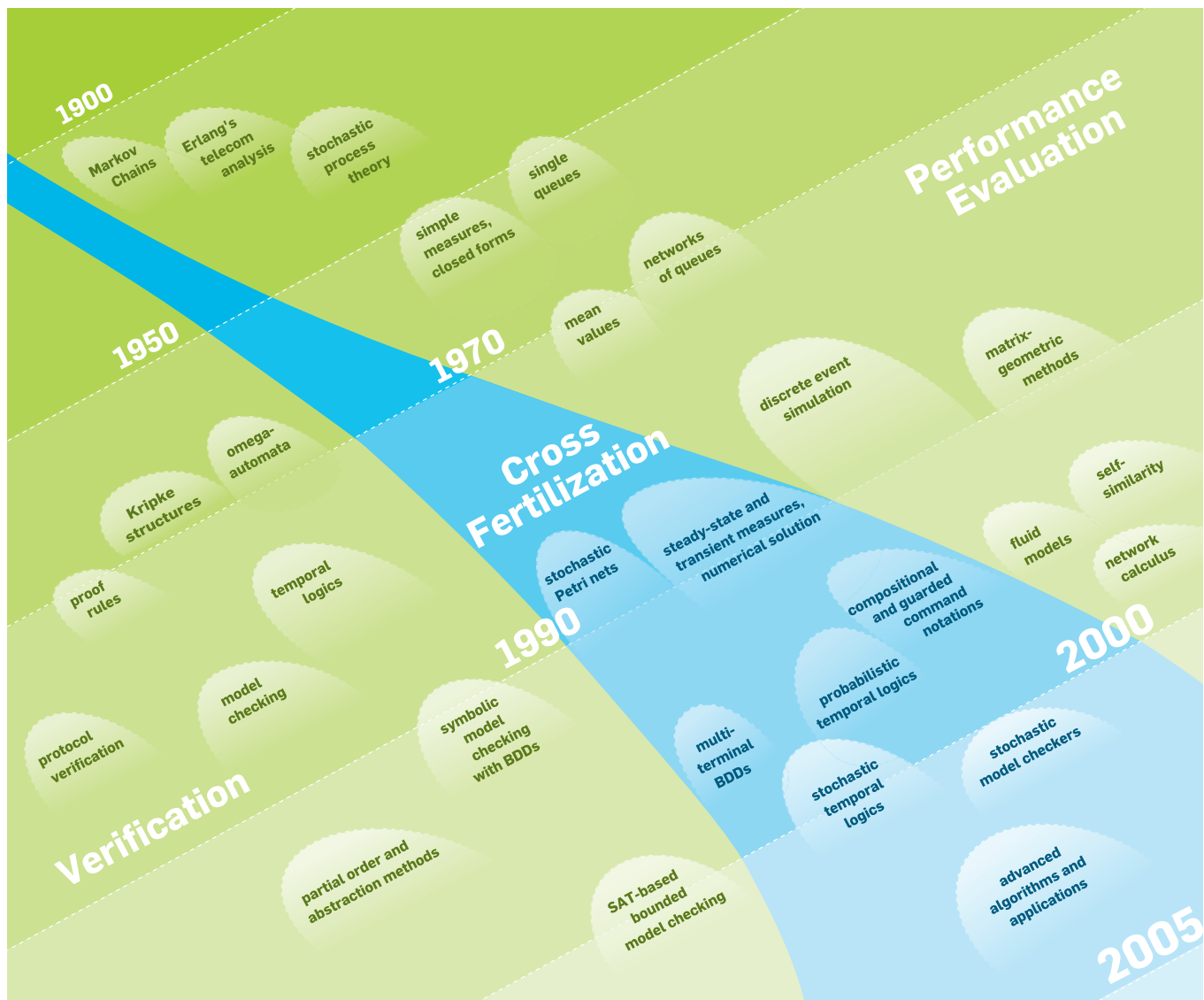
servers differ in computational capabilities? And what to do when multiple hosts have the same queue length? Well, the “join-the-shortest queue” policy might be adequate in most cases, but surely not in all. Its adequacy also depends on what quantity—or measure—one is interested in. This may be the mean delay of service requests, the mean queue length of waiting requests, rejection rates for waiting requests, and so on.

The effect of queue-selection policies on measures of interest or on decisions on how many servers are needed to reduce the waiting time by a given percentage, are answered by *performance evaluation* techniques. This branch of computer (system) science studies the perceived performance of systems based on an architectural system description and a workload model. Prominent techniques to obtain the aforementioned measures of interest are mathematical analysis that is typically focused on obtaining closed-form expressions, numerical evaluation that heavily relies on methods from linear algebra, and (discrete-event) simulation techniques that are based on statistical methods. The study and description of stochastic processes, most notably *Markov chains*, is pivotal for these techniques.

A complementary issue to performance is correctness. The central question is whether a system is conforming to the requirements and does not contain any flaws. Typically updates to our news Web site are queued, and it is relevant to know whether such

» key insights

- Performance engineers and verification engineers are currently facing very similar modeling and analysis challenges.
- A joint consideration is possible, practical, beneficial, and is supported by effective tools.
- Quantitative model checkers are applicable to a broad spectrum of applications ranging from sensor networks to security and systems biology.



Timeline log-scaled from 2010 backward.

buffers may overflow, giving rise to losing—perhaps headline—news items. Can such situations ever occur? Is there a possible scenario in which the dispatcher and application server are mutually waiting for each other, thus effectively halting the system? If such situations make the CNN news site unreachable on a presidential Election Day, this has far-reaching consequences. And what if the content of Web pages unexpectedly depend on the ordering of seemingly unrelated events in the application servers? Such “race conditions” should, if possible, be avoided.

A prominent discipline in computer science to assure the absence of errors, or, complementarily, to find errors (“bug hunting”) is *formal verification*. The spectrum of key techniques in this field ranges from runtime verification,

such as checking properties while executing the system, to deductive techniques such as theorem proving, to model checking. The latter is a highly automated model-based technique assessing whether a system model, that is, the possible system behavior, satisfies a property describing the desirable behavior. Typically, properties are expressed in temporal extensions of propositional logic, and system behavior is captured by Kripke structures, that is, finite-state automata with labeled states. Traditionally, such models do not incorporate quantitative information like timing or likelihoods of event occurrences.

The purpose of this article is to report on combining performance evaluation with model checking. Although these fields have been developed by different research communities in the past, over the last decade we have seen an integra-

tion of these two techniques for system analysis. Significant merits of this trend are a major increase of the applicability to real cases, and an impulse in the further development for both fields.

A Historic Account

To appreciate the benefits of combining performance evaluation and model checking, it is worthwhile to reflect on past and recent developments. We aim to shed light on the hidden assumptions associated with these developments. For more details on performance evaluation we refer to Bolch et al.¹⁰ and Jain,²² for details on model checking we refer to Baier and Katoen⁷ and Clarke et al.¹¹

Single queues. Performance evaluation dates back to the early 1900s, when Erlang developed models to dimension the number of required lines

in analogue telephone switches, based on the calculation of call loss probabilities. In fact, he used a queueing model, in which a potentially infinite supply of customers (callers) competes for a limited set of resources (the lines). The set of models and the theory that evolved from there is known as queueing theory. It has found, through the last century, wide applicability especially in telecommunications. Characteristic for most models is the competition for a single scarce resource at a time, leading to models with a single queue.

A large variety of modeling assumptions were made, for example, regarding the number of available servers (lines), buffering facilities, scheduling strategies, job discrimination, and the timing involved. The timings were assumed to follow some continuous-time distribution, most often a negative exponential distribution, leading to (what we now call) Markovian models. These models were subsequently analyzed, using calculus, to obtain such quantities as mean number of customers queued, mean delay, sometimes even the delay distribution, or the call blocking probability (“hearing a busy signal”). Many of these measures are available in closed form; at other times, numerical recipes were proposed, for example, to derive such measures from explicit expressions in the Laplace domain. Important to note is that model construction, as well as solution, was (and still is) seen as a craft, only approachable by experts.

Networks of queues. In the late 1960s, computer networks, networked computer systems, and time-sharing computer systems came into play. These systems have the distinguishing feature that they serve a finite customer population; however, they comprise multiple resources. This led to developments in the area of queueing networks, in which customers travel through a network of queues, are served at each queue according to some scheduling discipline, and are routed to their next point of service, and so on, until returning to the party that originated the request. Efficient algorithms to evaluate networks of queues to obtain a set of “standard measures” such as mean delays, throughputs, and mean queue lengths, were developed in the 1970s.^{40,51} A variety of software tools emerged support-

ing these algorithms that typically have a polynomial complexity in the number of queues and customers.

Stochastic Petri nets. In early 1980s, new computer architectures asked for more expressive modeling formalisms. In particular, parallel computers motivated modeling notions to spawn customers and to recombine smaller tasks into larger ones (fork/join queues). Moreover, the simultaneous use of multiple resources needed to be studied. Clearly, these concepts could not be expressed using queueing networks. This led to the proposal to extend Petri nets—originally developed to model concurrency—with a notion of time, leading to (generalized) stochastic Petri nets (SPNs).² Here, the tokens can either play the role of customers or of resources. Two observations are important. First, due to the increase in expressivity, specialized algorithms, such as those available for queueing networks, are typically no longer used. Instead, the SPN models must be mapped to an underlying stochastic process, a Markov chain that is solved by numerical means. Hence, the state space of the model must be generated explicitly, and the resulting Markov chain has to be solved numerically (linear equation solvers).

The computational complexity of these state-based methods is polynomial in the number of states, but this often is, in turn, (often) a high-degree polynomial in the SPN size. Secondly, as a result of the new solution trajectory, tool support became a central issue. Results achieved in this area also inspired new numerical algorithms for extended queueing network models. With hindsight, SPNs can be considered as the first “product” of the marriage between the field of performance evaluation and the field of formal modeling. In the 1990s, this trend continued and led to probabilistic variants of guarded command languages and of process algebras, the latter focusing on compositionality.

Nondeterminism. All of the models mentioned here are full stochastic models; that is, at no point in the model can some behavioral alternatives be left unspecified. For instance, the join-the-shortest-queue strategy leaves it open as to how to handle the case of several equally short queues.

This choice cannot be left open with the methods noted earlier; leaving such a choice is regarded as underspecification. What typically happens is that these cases are dealt with probabilistically, for example, by assigning probabilities to the alternatives. That is, nondeterminism is seen as a problem that must be removed before analysis can take place. This is important especially for modeling formalisms as SPNs; tools supporting the evaluation of these models will either detect and report such nondeterminism through a “well-specified check” or will simply insert probabilities to resolve it. In this case, analysis is carried out under a hidden assumption, and there is no guarantee that an actual implementation will exhibit the assumed behavior, nor that the performance derived on the basis of this assumption is achieved.

Trends. The last 20 years have seen a variety of developments in performance evaluation, mostly related to specific application fields, such as the works on effective bandwidth,⁴² network calculus,⁴⁶ self-similar traffic models,⁴⁷ and traffic (and mobility) models⁴³ (for communication network dimensioning purposes). A more general concept has been the development of fluid models to avoid the state space explosion problem (for example, Horton et al.⁴⁵) by addressing a large denumerable state space as a single continuous state variable. Furthermore, queueing network models have been extended with layering principles to allow for the modeling of software phenomena.⁵² Finally, work on matrix geometric methods⁴⁸ has led to efficient analysis methods for large classes of queueing models.

Model Checking

Proof rules. The fundamental question “when and why does software not work as expected?” has been the subject of intensive research since the early days of computer science. Software quality is typically based on peer review, such as manual code inspection, extensive simulation, and testing. These rather ad hoc validation techniques have severe limitations and restrictions. Research in the field of formal verification has led to complementary methods aimed at establishing software correctness with a very high level

of confidence. The origins of a sound mathematical approach toward program correctness—at a time where programs were described as flow diagrams—can be traced back to Turing in the late 1940s. Early attempts to assess the correctness of computer programs were based on mathematical proof rules that allow to reason in a purely syntax-based manner. In the 1960s, these techniques were developed for sequential programs, whereas about a decade later, this approach was generalized toward concurrent programs, in particular shared-variable programs.

Temporal logic. These syntax-based approaches are based on an interpretation of programs as input/output transformers and serve to prove partial correctness (such as soundness of output values for given inputs, provided the program terminates) and termination. Thanks to a key insight in the late 1970s by Pnueli, one recognized the need for concurrent programs to not only make assertions about the starting and final state of a program, but also about the states during the computation. This led to the introduction of temporal logic in the field of formal verification.⁵⁰ Proofs, however, were still conducted mainly by hand along the syntax of programs. Proofs for programs of realistic size, though, were rather lengthy and required a good dose of human ingenuity. In the field of communication protocols, the first techniques appeared toward automated checking of elementary properties.⁵³

Model checking. In the early 1980s, an alternative to using proof rules was proposed that checks systematically whether a (finite) model of a program satisfies a given property.^{7,11} The pioneers Clarke, Emerson, and Sifakis, received the ACM Turing Award 2007 for this breakthrough; it was the first step toward the fully automated verification of concurrent programs. How does model checking work? Given a model of the system (the possible behavior) and a specification of the property to be considered (the desirable behavior), model checking is a technique that systematically checks the validity of the property in the model. Models are typically nondeterministic finite-state automata, consisting of a finite set of states and a set of transi-



The strength of model checking is not in providing a rigorous correctness proof, but rather the ability to generate diagnostic feedback in the form of counterexamples in case a property is refuted.



tions that describe how the system evolves from one state into another. These automata are usually composed of concurrent entities and are often generated from a high-level description language such as Petri nets, process algebras, Promela, or Statecharts. Properties are specified in temporal logic such as Computation Tree Logic (CTL), an extension of propositional logic that allows one to express properties that refer to the relative order of events. Statements can either be made about states or about paths, such as sequences of states that model system evolution.

The backbone of the CTL model checking procedure is a recursive descent over the parse tree of the formula under consideration where temporal conditions (for example, a reachability for an invariance condition) are checked using fixed point computations. The class of path properties expressible in CTL is restricted to local conditions on the current states and its direct successors, constrained reachability conditions—is a goal state reachable by not visiting certain states before?—and their duals.

More complex path properties such as repeated reachability or progress properties, which, for example, can state that whenever a request enters the news Web site, it is served eventually, can be specified in Linear Temporal Logic (LTL). The rough idea of model checking LTL specifications is to transform the formula at hand into an automaton (recognizing infinite words) and then to analyze the product of this automaton with the system model by means of graph algorithms.

The strength of model checking is not in providing a rigorous correctness proof, but rather the ability to generate diagnostic feedback in the form of counterexamples (such as error traces) in case a property is refuted. This information is highly relevant to find flaws in the model and in the real system.

Taming state space explosion. The time and space complexity of these algorithms is linear in the size of the finite-state automaton describing the system. The main problem is this size may grow exponentially in the number of program and control variables, and in the number of components in a multithreaded or distributed system.

Since the birth of model checking, effective methods have been developed to combat this state explosion problem. Prominent examples of such techniques are: symbolic data structures,³⁹ partial-order reduction,⁴⁹ casting model checking as SAT-problems,³⁸ or abstraction techniques.¹¹ Due to these techniques, together with unre-

mitting improvements of underlying algorithms and data structures and hardware technology improvements, model checking techniques that only worked for simple examples a decade ago, are now applicable to more realistic designs. State-of-the-art model checkers can handle state spaces of about 10^9 states using off-the-shelf

technology. Using clever algorithms and tailored data structures, much larger state spaces (up to 10^{120} states⁴¹) can be handled for specific problems and reachability properties.

Quantitative aspects. From the early 1990s on, various extensions of model checking have been developed to treat aspects such as time and probabilities. Automata have been equipped with clock variables to measure the elapse of time (resulting in timed automata), and it has been shown that despite the infinite underlying state space of such automata, model checking of a timed extension of CTL is still decidable.³⁷ LTL has been interpreted over (discrete) probabilistic extensions of automata, focusing on the probability that an LTL formula holds, and probabilistic variants of CTL have been developed, as we will elaborate in more detail later on. For an overview, see Baier and Katoen.⁷ The combination of timing aspects and probabilities started about two decades ago and is highly relevant for this article.

Various software tools have been developed that support model checking. Some well-known model checking tools are: SPIN for LTL, NuSMV for CTL (and LTL), Uppaal for timed CTL, and PRISM for probabilistic CTL.

Let's Join Forces

Developments in performance evaluation lean toward more complex measures of interest, and focus on more complex system behavior. However, quantitative aspects such as timing and random phenomena are becoming more important in the field of model checking. Performance evaluation and model checking have thus grown in each other's direction, simply because from either end, it was felt that the methods in isolation did not answer the questions that were at stake. Let us discuss the reasons for this, and the benefits of combining these methods.

Individual Shortcomings

Why is a performance (or a dependability) evaluation of a system in itself not good enough? And why is a formal verification of a system insufficient to validate its usefulness? These questions are best answered by taking a simple system design example, for instance a reliable data transmission protocol such as

Table 1. Availability measures and their logical specification.

long-run	$\mathbb{L}_{\leq p}(up)$
instantaneous	$\mathbb{P}_{\leq p}(\Diamond^{[t,t]}up)$
conditional instantaneous	$\mathbb{P}_{\leq p}(\Phi \mathcal{U}^{[t,t]}up)$
interval	$\mathbb{P}_{\leq p}(\Box^{[t,t]}up)$
long-run interval	$\mathbb{L}_{\leq p}(\mathbb{P}_{\leq q}(\Box^{[t,t]}up))$
conditional interval long-run	$\mathbb{P}_{\leq p}(\Phi \mathcal{U}^{[t,t]} \mathbb{L}_{\leq q}(up))$

Figure 1. A logic for quantitative properties: syntax and semantics.

Let $\mathbb{X}$ be a general stochastic process, i.e., an indexed family $\{X(t) \mid t \in \mathbb{T}\}$ of random variables taking values in the set S . The index set $\mathbb{T}$ denotes the time domain of $\mathbb{X}$ and is either discrete ($\mathbb{T} = \mathbb{N}$) or continuous ($\mathbb{T} = \mathbb{R}$). We suppose that all states have positive probability under the initial distribution μ_{init} , i.e., $\mu_{\text{init}}(s) = \Pr_{\mathbb{X}}(X(0) = s) > 0$ for all states s . For event E , let $\Pr_{\mathbb{X},s}(E)$ denote the probability for E under the condition that s is the start state. Each state is labeled by a set of atomic propositions that can be viewed as state predicates.

Logical formulas (denoted by capital greek letters Φ, Ψ) are given by the grammar:

$$\Phi ::= a \mid \Phi \wedge \Psi \mid \neg \Phi \mid \mathbb{P}_{\leq p}(\Phi \mathcal{U}^I \Psi) \mid \mathbb{L}_{\leq p}(\Phi)$$

Here, a is an atomic proposition, $p \in [0, 1]$, $\leq \in \{\leq, \geq, >, <\}$ and $\mathbb{I}$ is a closed interval of $\mathbb{T}$. The semantics of this logic is defined inductively as follows:

$$s \models a \text{ iff state } s \text{ is labeled with atomic proposition } a$$

$$s \models \Phi \wedge \Psi \text{ iff } s \models \Phi \text{ and } s \models \Psi$$

$$s \models \neg \Phi \text{ iff } s \not\models \Phi$$

$$s \models \mathbb{P}_{\leq p}(\Phi \mathcal{U}^I \Psi) \text{ iff } \Pr_{\mathbb{X},s}(\exists t \in \mathbb{I} (X(t) \models \Psi \wedge \forall t' \in \mathbb{T} (t' < t \Rightarrow X(t') \models \Phi)) \leq p$$

$$s \models \mathbb{L}_{\leq p}(\Phi) \text{ iff } \text{LRA}(s, \text{Sat}_{\mathbb{X}}(\Phi)) \leq p$$

where $\text{Sat}_{\mathbb{X}}(\Phi) = \{s \in S \mid s \models \Phi\}$ and for $B \subseteq S$, $\text{LRA}(s, B)$ denotes the "long run average" of being in a state of B for runs starting in state s . Formally, $\text{LRA}(s, B)$ is the expected value of the random variable

$$\lim_{t \rightarrow \infty} \frac{1}{t} \int_0^t \mathbf{1}_B(X(\theta)) d\theta$$

with respect to the probability measure $\Pr_{\mathbb{X},s}$. Here, $\mathbf{1}_B$ denotes the characteristic function of B , i.e., $\mathbf{1}_B(s') = 1$ if $s' \in B$ and 0 otherwise.

Derived operators. Let $\mathcal{U}^I$ denote $\mathcal{U}$. Usual propositional operators such as **ff**, **tt**, $\vee$ are derivable. The eventually operator $\Diamond^I$ with time bounds given by a time interval $\mathbb{I}$ is obtained by $\Diamond^I \Phi = \text{tt} \mathcal{U}^I \Phi$. To specify that condition Φ holds continuously in the time interval $\mathbb{I}$, the time-constrained always operator $\Box^I$ can be defined by using the duality of "eventually" and "always". For instance, $\mathbb{P}_{\geq p}(\Box^I \Phi)$ is a shorthand notation for $\mathbb{P}_{\leq 1-p}(\Diamond^I \neg \Phi)$.

TCP. Such a protocol relies on a number of ingredients that, when suitably combined, result in the desired behavior: reliable, end-to-end in-order delivery of packets between communicating peers. These ingredients comprise timers, sequence numbers, retransmissions, and error-detecting codes.

A typical performance model will take into account the TCP timing and retransmission aspects, whereas the error correction will mostly be included as a random phenomenon.

For the sake of simplicity, sequence numbers are neglected, which results in a model that can be analyzed using either a closed-form formula or some numerical technique that, under the assumption the model is functionally correct, gives a certain mean performance, measured as throughput or mean packet delay. However, the obtained quantities do not say anything about the question of whether the packets do arrive correctly at all, hence, whether the protocol is correct. Conversely, a classical functional model of the sketched protocol here will most likely result in a correctness statement of the form “all packets will eventually arrive correctly.” But this gives no information about perceived delays and throughputs. Needless to say, one cannot simply “add up” the results of both analyses, as they result from two different—and possibly quite unrelated—models.

The key challenge lies in developing an integrated model. Preferably, the user, such as the system architect or design engineer, just provides a single model (as engineering artifact) that forms the basis for both types of analysis. To improve the efficiency, additional property-dependent abstraction techniques can be applied to abstract away from all details of the model that are irrelevant for the property to be checked. For example, checking whether a purely functional property holds for a Markov model requires an analysis of the underlying graph structure, and one can ignore all stochastic information.

Benefits

Modeling and measure specification. An important advantage of using temporal logics (or automata) to specify properties of interest—in fact guarantees on measures of interest—is the

possibility to describe properties at the same abstraction level as the modeling of the stochastic process. Up to now, it has been tradition to specify measures of interest such as “what is the probability to fail within deadline d ?” at state level, that is, in terms of the states and their elementary properties (logically speaking, atomic propositions). Sometimes reward structures have been added at state level to quantify the use of resources such as queue occupancies and the like. This stands in sharp contrast with the description of the models themselves, which is mostly done using high-level modeling formalisms such as queueing networks, SPNs, stochastic automata networks, or stochastic process algebra. Temporal logics close this paradigm gap between high-level and state-based modeling as they allow to specify properties in terms of the high-level models, for example, in terms of the token distribution among places in a

Petri net. By the use of temporal logics, modeling and measure specification become treated at an equal footing.

An example logic with semantics interpretation is illustrated in Figure 1. Instances of this generic logic arise by considering special types of stochastic processes, for example, for an interpretation over discrete-time Markov chains (DTMC), $T = N$ and we obtain probabilistic computation tree logic (PCTL).¹⁸ For continuous-time Markov chains (CTMC), the time domain is $T = R$, and continuous stochastic logic (CSL) is obtained.^{4,6} Figure 3 presents a small representative example^{3,9} with some typical logical formulae.

Expressivity and flexibility. The use of logics offers, in addition, a high degree of expressiveness. Simple performance and dependability metrics such as transient probabilities—what is the probability of being in a failure state at time t ?—and long-run likelihoods (when

Figure 2. Schema for model checking stochastic processes.

given: a stochastic process $\mathbb{X}$ and a logical formula Φ

task: compute $\Pr_{\mathbb{X}}\{X(0) \models \Phi\}$

idea: compute the sets $Sat_{\mathbb{X}}(\Psi) = \{s \in S \mid s \models \Psi\}$ for any subformula Ψ of Φ and return $\sum_{s \in Sat_{\mathbb{X}}(\Phi)} \mu_{init}(s)$

► $Sat_{\mathbb{X}}(a) = \{s \in S \mid \text{state } s \text{ is labeled with atomic proposition } a\}$

► $Sat_{\mathbb{X}}(\Psi_1 \wedge \Psi_2) = Sat_{\mathbb{X}}(\Psi_1) \cap Sat_{\mathbb{X}}(\Psi_2)$

► $Sat_{\mathbb{X}}(\neg\Psi) = S \setminus Sat_{\mathbb{X}}(\Psi)$

► computation of $Sat_{\mathbb{X}}(\mathbb{P}_{\leq p}(\Psi_1 \mathcal{U}^i \Psi_2))$:

case 1: $\mathbb{I} = [0, t]$ for some $t \in \mathbb{T}$, $t > 0$. Let $\mathbb{Y}$ be the stochastic process that results from $\mathbb{X}$ by making all states where Ψ_2 holds or Ψ_1 is refuted absorbing. That is, if $B = Sat_{\mathbb{X}}(\Psi_2) \cup S \setminus Sat_{\mathbb{X}}(\Psi_1)$, then $\mathbb{Y}$ is given by

$$Y(t) = \begin{cases} X(t) & \text{if } X(t') \notin B \text{ for all } t' < t \\ s & \text{if } X(t') = s \in B \text{ for some } t' < t \text{ and } X(t'') \notin B \text{ or } X(t'') = s \text{ for all } t'' < t'. \end{cases}$$

Apply known methods of performance evaluation to compute the probabilities

$$p_s = \Pr_{\mathbb{Y},s} \{X(t) \in Sat_{\mathbb{X}}(\Psi_2)\}$$

and return $Sat_{\mathbb{X}}(\mathbb{P}_{\leq p}(\Psi_1 \mathcal{U}^i \Psi_2)) = \{s \in S \mid p_s \leq p\}$.

case 2: $\mathbb{I} = [t_1, t_2]$ for some $t_1 > 0$. Let $\mathbb{Y}$ be the stochastic process that arises from $\mathbb{X}$ by making all states refuting Ψ_1 absorbing. Regard the stochastic process $\mathbb{Z}$ that arises from $\mathbb{Y}$ by shifting the time by t_1 time units, i.e., $\mathbb{Z}$ is specified by $Z(t) = Y(t + t_1)$. We then evaluate the formula $\mathbb{P}_{\leq p}(\Psi_1 \mathcal{U}^{[0, t_2 - t_1]} \Psi_2)$ over $\mathbb{Z}$ as in case 1 and return

$$Sat_{\mathbb{X}}(\mathbb{P}_{\leq p}(\Psi_1 \mathcal{U}^i \Psi_2)) = Sat_{\mathbb{Z}}(\mathbb{P}_{\leq p}(\Psi_1 \mathcal{U}^{[0, t_2 - t_1]} \Psi_2)).$$

► Let $B = Sat_{\mathbb{X}}(\Phi)$ and apply known methods of performance evaluation to compute the long run average $LRA(s, B)$ of being in a state of B for runs starting in state s . Return

$$Sat_{\mathbb{X}}(\mathbb{L}_{\leq p}(\Phi)) = \{s \in S \mid LRA(s, B) \leq p\}.$$

the system is observed long enough) can readily be expressed. Most standard performance measures are easily captured, see Table 1 for a selection of properties. More importantly, the use of logics offers an enormous degree of flexibility. Nesting formulas yields a simple mechanism to specify complex measures in a succinct manner. A property like “the probability to reach a state within 25 seconds that almost surely stays safe for the next 10 seconds, via legal states only exceeds $\frac{1}{2}$ ” boils down to

$$\mathbb{P}_{>\frac{1}{2}}(\text{legal } U \leq^{25} \mathbb{P}_{=1}(\Box \leq^{10} \text{safe}))$$

This immediately pinpoints another advantage: given the formal semantics of the temporal logic, the meaning of the above formula is precise. That is to say, there is no possibility that any confusion might arise about its meaning. Unambiguous measure specifications are of utmost importance. Existing mathematical measure specifications are rigorous too of course, but do not offer the flexibility and succinctness

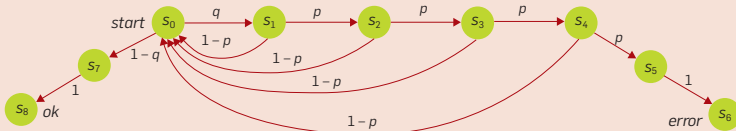
of logics. Temporal logic provides a framework that is based on just a few basic operators.

Many measures, one algorithm. The above concerns the measure specification. The main benefit though is the use of model checking as a fully algorithmic approach toward measure evaluation. Even better, it provides a single computational technique for any possible measure that can be written. This applies from simple properties to complicated, nested, and possibly hard-to-grasp formulas. For the example logic this is illustrated in Figure 2. This is radically different from common practice in performance and dependability evaluation where tailored and brand new algorithms are developed for “new” measures. One might argue that this will have a high price, that is, the computational and space complexity of the exploited algorithms must be extremely high. No! On the contrary, in the worst case, the time complexity is linear in the size of the measure specification (logic formula), and polynomial (typically of order 2 or 3, at most) in the number of states of the stochastic process under consideration. As indicated in Figure 4, the verification of bounded reachability probabilities in DTMCs and CTMCs—often the most time-consuming ones—is a matter of a few seconds even for millions of states: The space complexity is quadratic in the number of states in the worst case. In fact, as for other state-based performance evaluation techniques this polynomial complexity is an issue of concern as the number of states may grow rapidly.

Perhaps the largest advantage of model checking for performance analysis is that all algorithmic details, all detailed and non-trivial numerical computation steps are hidden to the user. Without any expert knowledge on, say, numerical analysis techniques for CTMCs, measure evaluation is possible. Even better: the algorithmic analysis is measure-driven. That is to say, the stochastic process can be tailored to the measure of interest prior to any computation, avoiding the consideration of parts of the state space that are irrelevant for the property of interest. In this way, computations must be carried out only on the fragments of

Figure 3. A simple model checking example: The Zeroconf protocol.

The IPv4 zeroconf protocol is a simple protocol proposed by the IETF (RFC 3927), aimed at the self-configuration of IP network interfaces in ad hoc networks. Such ad hoc networks must be hot-pluggable and self-configuring. Among others, this means that when a new appliance, hitherto called a newcomer, is connecting to a network, it must be configured with a unique IP address automatically. The zeroconf protocol solves this task using randomization. A newcomer intending to join an existing network randomly selects an IP address, U say, out of the 65024 available addresses and broadcasts a message (called a probe) asking “Who owns the address U ?”. If an owner of U is present and does receive that message, it replies, to force the newcomer to randomly select another address. Due to message loss or busy hosts, messages may not arrive at some hosts. Therefore a newcomer is required to send a total of four probes, each followed by a listening period of two seconds before it may assume that a selected address is unused. Therefore, the newcomer can start using the selected IP address only after eight seconds. Notably, there is a low probability risk that a newcomer may still end up using an already owned IP address, for example, because all probes were lost. This situation, called address collision, is highly undesirable.



The protocol behavior of a newcomer is easily modeled by a DTMC depicted above consisting of nine states.^{3,9} The protocol starts in s_0 where the newcomer randomly chooses an IP address. With probability $q = m/65024$ the address is already owned, where m is the current size of the network. State s_i ($0 < i \leq 4$) is reached after issuing the i -th probe. With probability p no reply is received during two seconds on a sent probe (as either the probe or its reply has been lost). State s_8 (labeled *ok*) indicates that eventually a unique address has been selected, while state s_6 (labeled *error*) corresponds to the undesirable situation of an address collision.

For such a model some typical example formulae are:

- On the long run, the protocol will have selected an address: $\mathbb{L}_{\geq 1}(ok \vee error)$.
- The probability to end up with an address collision is at most p : $\mathbb{P}_{\leq p'}(\Diamond error)$
- The probability to arrive at an unused address within k steps exceeds p' : $\mathbb{P}_{\leq p'}(\Diamond^{[0,k]} ok)$

Many more measures including expected times and accumulated costs can be expressed using extensions of the base logic and model introduced here.

the state space that are relevant to the property of interest. In fact, this generalizes the ideas put forward by Sanders and Meyer on variable-driven state space generation in the late 1980s.³³

Dependability evaluation. This measure-driven aspect is even more beneficial in the field of system dependability evaluation, a field tightly related to performance evaluation, but especially concerned with evaluating service continuity of computer systems. Questions like “under which system faults can a given service still be provided adequately?” are addressed, and typical measures of interest are system reliability and availability, as illustrated in Table 1. Since the beginning of the 1980s this field has matured significantly, due to the introduction of state-oriented models and the invention of uniformization.⁴⁴ This facilitated the efficient analysis of time-dependent properties such as reliability or availability evaluation, in combination with high-level model specification techniques such as SPNs. The models that one could analyze now went well above the “standard models” based on reliability block diagrams or fault-trees.

The measures of interest in this field often involve costs, modeling the usage of resources. Extensions of stochastic processes with cost (or reward) functions give rise to a logic where in addition to, for example, time bounds, conditions about the accumulated reward along an execution path can be imposed. Model checking still goes along the lines of Figure 2, but involves computational procedures that are more time-consuming.

One for free. Is that all? Not quite. An important problem with performance modeling regardless of whether one aims at numerical evaluation or at simulation, is to check the functional correctness of the model. For a stochastic Petri net specification, place and transition invariants are exploited to check for deadlocks and liveness, among others.

For a Markov chain model, graph-based algorithms are used to check elementary properties. The good news is when employing model checking we get this functionality for free. Using the same machinery for validating the measures of interest, functional properties can be checked. Probabilistic model

checking provides two for the price of one: both performance/dependability analysis and checking functional properties. This forces the user to construct models with a high precision as any tiny inconsistency will be detected. Compare this to simulation model construction in NS2 or OPNET!

Nondeterminism. Sometimes this need for precision might seem as a burden, but it is a vehicle to force the modeler to make hidden assumptions explicit—or to leave them out. For instance, we have discussed the nondeterminism inherent in the join-the-shortest-queue idea, which—unless made concrete—implies that the underlying model is not a stochastic process. Stochastic models with nondeterminism are usually referred to as stochastic decision processes. In these models the future behavior is not always determined by a unique probability distribution, but by selecting one from a set of them. Temporal logics and verification technology have been extended to this type of models with relative ease for CTL⁸ and LTL.^{14,36} In fact, they constitute the genuine supermodel that comprises both the model checking and performance evaluation side as special cases: When transition systems are paired with Markov chains or Markov reward models, the model is known as Markov decision processes. Here, performance model checking is still pos-

sible, but the checker now computes bounds on the performance, in the sense that however the nondeterminism is concretized, the concrete performance figure will stay within the calculated bounds. Whereas for the discrete time setting, efficient model checking algorithms have been developed, this field is still relatively open in the continuous-time setting.

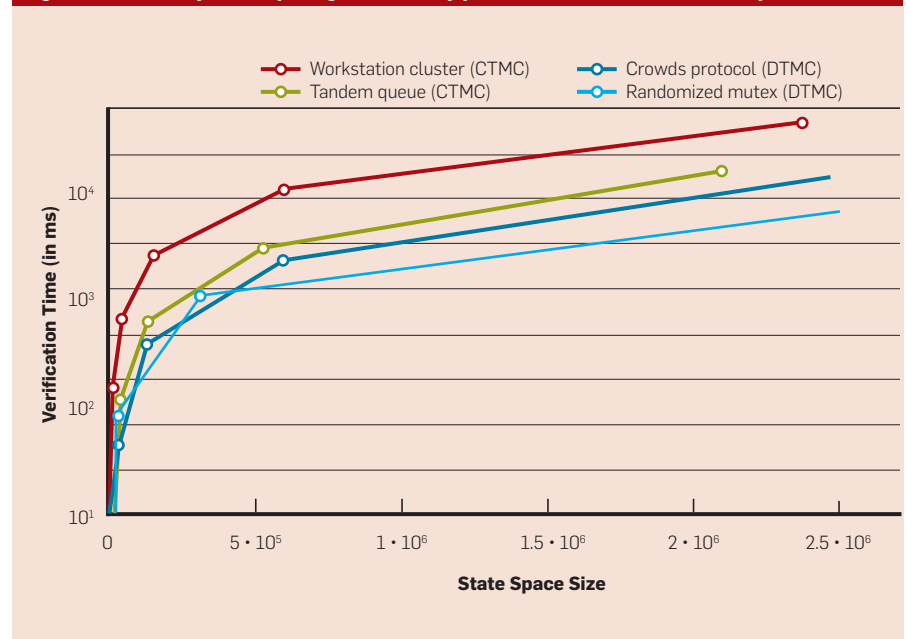
Appealing Application Areas

Several stochastic model checking tools have been developed since 2005, of which PRISM²⁰ is by far the most widely used. A number of well-known tools from the performance and dependability evaluation area, like tools for SPNs and stochastic process algebras, have been extended with stochastic model checking features. All these tools automatically generate a Markovian model of some sort, either using symbolic or sparse data structures.

With these tools, a wide variety of case studies have been carried out, amongst others, in application areas such as communication systems and protocols, embedded systems, systems biology, hardware design, and security, as well as more “classical” performance and dependability studies.

Examples of the latter category, for which CTMCs are a very natural model, include the analysis of various classes of traditional queuing networks and even infinite-state variants

Figure 4. Efficiency of computing reachability probabilities versus the state space size.



thereof, fault-tolerant workstation clusters, and wireless access protocols such as IEEE 802.11. Also system survivability, that is the ability of a system (for example, military or aircraft) to recover predefined service levels in a timely manner after the occurrence of disasters, has been precisely captured using a logic similar to that introduced before, and has been verified for Google-like file systems.¹² The evaluation of a wireless access protocol for ad hoc networks using model checking could be carried out at far lower cost than using discrete-event simulations.³²

The popularity of Markovian models is rapidly growing due to their application potential in systems biology; the timing and probabilistic nature of CTMCs naturally reflect the operations of biological mechanisms such as molecular reactions. In fact, various biological systems have been studied by CTMC model checking in recent years.²⁶ Prominent examples include ribosome kinetics, signaling pathways, cell cycle control in Eukaryotes, and enzyme-catalyzed substrate conversion. In particular, the possibility to compute time-bounded reachability probabilities is of great importance here as traditional studies focus on steady-state behavior.

Another application area for CTMC model checking is embedded systems where the timeliness of communication between sensor and actuator devices, for example, within cars or between high-speed trains, is of utmost importance. Stochastic model checking techniques allow us to address the timeliness and the protocols' correctness from a single model. One example is dynamic power management in relation to job scheduling.³¹

Examples for the discrete-time setting include several studies of the IPv4 Zeroconf protocol are illustrated in Figure 3, where next to the probability of eventually obtaining an unused IP address, extensions have been studied with costs, addressing issues such as the number of attempts needed to obtain such address. Security protocols are another important class of systems in which discrete randomness is exploited, for example, by applying random routing to avoid information leakage. An interesting case is the



An important problem with performance modeling regardless whether one aims at numerical evaluation or at simulation, is to check the functional correctness of the model.



Crowds protocol,³⁴ a well-known security protocol that aims to hide the identity of Web-browsing stations. Checking Markovian models with up to 10^7 states did provide important information on quantifying the increase of confidence of an adversary when observing an Internet packet of the same sender more than once. A novel case study in the field of nanotechnology applies stochastic model checking to quantify the reliability of a molecular switch with increasing memory array sizes.¹³ Other natural cases for discrete-time probabilistic models are randomized protocols—in which probabilities are used to break ties—such as consensus and broadcast protocols, and medium access mechanisms such as Zigbee.

To conclude, an interesting case study using DTMCs with non-determinism is the analysis of the Firewire protocol (IEEE 1394). This protocol has been developed to allow “plug-and-play” network connectivity for multimedia consumer electronics in the home environment. A key component in IEEE 1394 is a leader election protocol (the “root contention protocol”) that exploits a coin-tossing mechanism to break ties. Stochastic model checking revealed that using a biased coin instead of the typically used unbiased coin, speeds up the leader election process. This confirmed a conjecture in Stoelinga.³⁵ This insight would not have been found through “classical” qualitative verification.

Current Trends and Challenges

Edmund M. Clarke, a co-recipient of the 2007 ACM A.M. Turing Award, points out that probabilistic model checking is one of the brands of verification that requires further developments.⁴¹ Here, we note some of the current trends and major research challenges.

One of the major practical obstacles shared by model-based performance evaluation and model checking is the state space explosion problem. To combat the state space explosion problem, various techniques have been developed and successfully applied for model checking Kripke structures¹¹ (and the literature mentioned there).

For stochastic models the state space explosion problem is even more severe. This is rooted in the fact that

the model checking algorithms for stochastic models rely on a combination of model checking techniques for non-stochastic systems, such as graph algorithms, but also mathematical, often numerical methods for calculating probabilities, such as linear equation solving or linear programming.

Many of the advanced techniques for very large non-stochastic models have been adapted to treat stochastic systems, including variations of decision diagrams to represent large state spaces symbolically.³⁰ Complementary techniques attempt to abstract from irrelevant or redundant details in the model and to replace the model with a smaller, but “equivalent” one. Some of them rely on the concept of lumpability for stochastic processes, which in the formal verification setting is known as bisimulation quotienting, and where states with the same probabilistic behavior are collapsed into a single representative.^{16,27}

Other advanced techniques to fight the state-explosion problem include symmetry exploitation,²⁴ partial order reduction,⁵ or some form of abstraction²⁸, possibly combined with automatic refinement.^{15,19} All these approaches take inspiration in classical model checking advances, which often get much more intricate to realize, and raise interesting theoretical and practical challenges. All together, they have advanced the field considerably in the ability to handle cases as the ones discussed earlier.

An important feature of model checkers for non-stochastic systems is the generation of counterexamples for properties that have been refuted by the model checker. The principal situation is more difficult in the stochastic setting, as for probabilistic properties, say the requirement that a certain undesired event will appear with probability at most 10^{-3} , single error traces are not adequate. The generation and representation of counterexamples is therefore a topic of much increasing attention^{17,29} within the community.

To overcome the limitation to finite state spaces, much work has been done to treat infinite-state probabilistic systems, in many different flavors.^{1,23}

Another topic of ongoing interest lies in combining probabilistic behavior with continuous dynamics as in

timed²⁵ or hybrid automata, but more work on the tool side is needed to assess the merits of these approaches faithfully. Theorem-proving techniques for analyzing probabilistic systems²¹ are also a very promising direction. One of the major open technical problems is the treatment of models with nondeterminism and continuous distributions. Initial results are interesting but typically subject to (severe) restrictions.

As a final item, we mention the need to tailor the general-purpose probabilistic model checking techniques to special application areas. This covers the design of special modeling languages and logics that extend or adapt classical modeling languages and temporal logics by adding features that are specific for the application area.

Acknowledgments

We thank Andrea Bobbio, Gianfranco Ciardo, William Knottenbelt, Marta Kwiatkowska, Evgenia Smirni, and the anonymous reviewers for their valuable feedback. C

References

Due to space limitations, a comprehensive list of all references cited in this article can be found at the authors' Web sites.

- Abdulla, P., Bertrand, N., Rabinovich, A. and Schnoebelen, P. Verification of probabilistic systems with faulty communication. *Inf. and Comp.* 202, 2 (2007), 141–165.
- Ajmani Marsan, M., Conte, G., Balbo, G. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comput. Syst.* 2, 2 (1984), 93–122.
- Andova, S., Hermanns, H., and Katoen, J.-P. Discrete-time rewards model-checked. *FORMATS, LNCS 2791*, (2003), 88–104.
- Aziz, A., Sanwal, K., Singhal, V. and Brayton, R.K. Model checking continuous-time Markov chains. *ACM TOCL* 1, 1 (2000), 162–170.
- Baier, C., Gröber, M., and Ciesinski, F. Partial order reduction for probabilistic systems. *Quantitative Evaluation of Systems*. IEEE CS Press, 2004, 230–239.
- Baier, C., Haverkort, B.R., Hermanns, H., and Katoen, J.-P. Model checking algorithms for continuous-time Markov chains. *IEEE TSE* 29, 6 (2003), 524–541.
- Baier, C. and Katoen, J.-P. *Principles of Model Checking*. MIT Press, 2008.
- Bianco, A. and De Alfaro, L. Model checking of probabilistic and non-deterministic systems. *Foundations of Software Technology and Theor. Comp. Science. LNCS 1026* (1995), 499–513.
- Bohnenkamp, H., van der Stok, P., Hermanns, H., and Vaandrager, F.W. Cost optimisation of the IPv4 zeroconf protocol. In *Proceedings of the Intl. Conf. on Dependable Systems and Networks*. IEEE CS Press, 2003, 531–540.
- Bolch, G., Greiner, S., de Meer, H., Trivedi, K.S. *Queueing Networks and Markov Chains*. Wiley Press, 1998.
- Clarke, E.M., Grumberg, O., and Peled, D. *Model Checking*. MIT Press, 1999.
- Cloth, L. and Haverkort, B.R. Model checking for survivability. *Quantitative Evaluation of Systems*. IEEE CS Press (2005), 145–154.
- Coker, A., Taylor, V., Bhaduri, D., Shukla, S., Raychowdhury, A., and Roy, K. Multijunction fault-tolerance architecture for nanoscale crossbar memories. *IEEE Trans. on Nanotechnology* 7, 2 (2008), 202–208.
- Courcoubetis, C. and Yannakakis, M. The complexity of probabilistic verification. *JACM* 42, 4 (1995), 857–907.
- D'Argenio, P.R., Jeannot, B., Jensen, H., and Larsen, K.G. Reduction and refinement strategies for probabilistic analysis. *LNCS 2399* (2002), 335–372.
- Derisavi, S., Hermanns, H., and Sanders, W.H. Optimal state-space lumping in Markov chains. *Inf. Proc. Letters* 87, 6 (2003), 309–315.
- Han, Y., Katoen, J.-P. and Damman, B. Counterexamples in probabilistic model checking. *IEEE TSE* 36, 3 (2010), 390–408.
- Hansson, H. and Jonsson, B. A logic for reasoning about time and reliability. *Formal Aspects of Comp.* 6, 5 (1994), 512–535.
- Hermanns, H., Wachter, B., and Zhang, L. Probabilistic CEGAR. *Computer-Aided Verification LNCS 5123* (2008), 162–175.
- www.prismmodelchecker.org.
- Hurd, J., McIver, A., and Morgan, C. Probabilistic guarded commands mechanized in HOL. *Theor. Comp. Sc.* 346, 1 (2005), 96–112.
- Jain, R. *The Art of Computer System Performance Analysis*. Wiley, 1991.
- Kucera, A., Esparza, J., and Mayr, R. Model checking probabilistic pushdown automata. *Logical Methods in Computer Science* 2, 1 (2006).
- Kwiatkowska, M.Z., Norman, G., and Parker, D. Symmetry reduction for probabilistic model checking. *Computer-Aided Verification LNCS 4144* (2008), 238–248.
- Kwiatkowska, M.Z., Norman, G., Parker, D., and Sproston, J. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design* 29, 11 (2006), 33–78.
- Kwiatkowska, M.Z., Norman, G., and Parker, D. Probabilistic model checking for systems biology. *Symbolic Systems Biology*, 2010.
- Larsen, K.G. and Skou, A. Bisimulation through probabilistic testing. *Inf. & Comp.*, 94, 1 (1989), 1–28.
- McIver, A. and Morgan, C. *Abstraction, Refinement and Proof for Probabilistic Systems*. Springer, 2005.
- McIver, A., Morgan, C., and Gonzalia, C. Proofs and refutation for probabilistic systems. *Formal Methods LNCS 5014* (2008), 100–115.
- Miner, A. and Parker, D. Symbolic representation and analysis of large probabilistic systems. *Validation of Stochastic Systems. A Guide to Current Research. LNCS 2925* (2005), 296–338.
- Norman, G., Parker, D., Kwiatkowska, M.Z., Shukla, S.K., Gupta, R. Using probabilistic model checking for dynamic power management. *Formal Asp. Comp.* 17, 2 (2005), 160–176.
- Remke, A., Haverkort, B.R. and Cloth, L. A versatile infinite-state Markov reward model to study bottlenecks in 2-hop ad hoc networks. *Quantitative Evaluation of Systems IEEE CS Press*, 2006, 63–72.
- Sanders, W.H. and Meyer, J.F. Reduced base model construction methods for stochastic activity networks. *IEEE J. on Selected Areas in Comms.* 9, 1 (1991), 25–36.
- Shmatikov, V. Probabilistic model checking of an anonymity system. *J. Computer Security* 12, (2004) 355–377.
- Stoelinga, M. Fun with FireWire: A comparative study of formal verification methods applied to the IEEE 1394 root contention protocol. *Formal Asp. Comp.*, 14, 3 (2003), 328–337.
- Vardi, M.Y. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of the 26th IEEE Symp. on Foundations of Comp. Science*. IEEE CS Press (1985), 327–338.

Christel Baier (baier@tcs.inf.tu-dresden.de) is a professor at TU Dresden, Germany.

Boudewijn R. Haverkort (brh@cs.utwente.nl) is a professor at the University of Twente, and scientific director of the Embedded Systems Institute, Eindhoven, The Netherlands.

Holger Hermanns (hermanns@cs.uni-sb.de) is a professor at Saarland University, Saarbrücken, Germany.

Joost-Pieter Katoen (katoen@cs.rwth-aachen.de) is a professor at RWTH Aachen University, Aachen, Germany.



Association for
Computing Machinery

Advancing Computing as a Science & Profession



You've come a long way. Share what you've learned.



ACM has partnered with MentorNet, the award-winning nonprofit e-mentoring network in engineering, science and mathematics. MentorNet's award-winning **One-on-One Mentoring Programs** pair ACM student members with mentors from industry, government, higher education, and other sectors.

- Communicate by email about career goals, course work, and many other topics.
- Spend just **20 minutes a week** - and make a huge difference in a student's life.
- Take part in a lively online community of professionals and students all over the world.



Make a difference to a student in your field.

Sign up today at: www.mentornet.net

Find out more at: www.acm.org/mentornet

MentorNet's sponsors include 3M Foundation, ACM, Alcoa Foundation, Agilent Technologies, Amylin Pharmaceuticals, Bechtel Group Foundation, Cisco Systems, Hewlett-Packard Company, IBM Corporation, Intel Foundation, Lockheed Martin Space Systems, National Science Foundation, Naval Research Laboratory, NVIDIA, Sandia National Laboratories, Schlumberger, S.D. Bechtel, Jr. Foundation, Texas Instruments, and The Henry Luce Foundation.

research highlights

P. 88

**Technical
Perspective
Programming with
Differential Privacy**

By Johannes Gehrke

P. 89

**Privacy Integrated Queries:
An Extensible Platform for
Privacy-Preserving Data Analysis**

By Frank McSherry

P. 98

**Technical
Perspective
Constraint
Satisfaction
Problems and
Computational
Complexity**

By Mark Jerrum

P. 99

**Constraint Satisfaction
Problems and Global
Cardinality Constraints**

By Andrei A. Bulatov and Dániel Marx

Technical Perspective

Programming with Differential Privacy

By Johannes Gehrke

GOVERNMENT AGENCIES WORLDWIDE ARE required to release statistical information about population, education, and health, crime, and economic activities. In the U.S., protecting this data goes back to the 19th century when Carrol Wright, the first head of the Bureau of Labor Statistics, which was established in 1885, argued that protecting the confidentiality of the Bureau's data was necessary. If enterprises feared that data about an enterprise collected by the Bureau would be shared with competitors, investigators, or the tax authorities, data quality would severely suffer. The field of statistical disclosure limitation was born.⁴

Fast-forward a few decades, Stanley Warner was faced with a similar conundrum. During interviews for market surveys, individuals would refuse questions of sensitive or controversial issue “for reasons of modesty, fear of being thought bigoted, or merely a reluctance to confide secrets to strangers.”⁷ His answer was a technique where the interviewee would flip a biased coin without showing the outcome to the interviewer. Depending on the outcome of the coin flip, the interviewee would (truthfully) answer either the original yes/no question or she would negate her answers. This method intuitively protects the interviewee since her answer could always have been due to the coin flipping on the other side.

Tore Dalenius formulated a very strong notion of protection a decade later:² “If the release of the statistic S makes it possible to determine the (microdata) value more accurately than without access to S , a disclosure has taken place...”. This very strong notion of semantic security implies that data publishers should think about adversaries and their knowledge since the published data could give new information to an adversary.

Fast-forward a few more decades to the turn of the century. Statisticians

have developed many different methods for limiting disclosure when publishing data such as suppression, sampling, swapping, generalization (also called coarsening), synthetic data generation, data perturbation, and the publishing of marginals for contingency tables, just to name a few. These methods are applied in practice, but they do not provide formal privacy guarantees—the methods do not formally state how much an attacker can learn, and they preserve confidentiality by hiding the parameters used.

Fast-forward to 1999. In his Innovations Award Talk at the annual ACM SIGKDD Conference, Rakesh Agrawal posed the challenge of privacy-preserving data mining to the community. In the next year, two papers with the same title “Privacy Preserving Data Mining” (one by Agrawal and Srikant¹ and the other by Lindell and Pinkas³) are published, and the computer science community has entered the picture.

Computer scientists were especially intrigued by formal models of data privacy—formal definitions of information leakage and attacker models as they have been pioneered and used in cryptography and computer security. The strongest formal definition of disclosure in use today is differential privacy as pioneered by Dwork, McSherry, Nissim, and Smith.³ Differential privacy beautifully captures the intuitive notion that the published data should not reveal much information about an individual whether or not that individual's data was in the data.

Since its original proposal, much progress has been made in the development of mechanisms that protect published data with differential privacy while maximizing information content. The national statistical offices have also started to pay attention; for example, OnTheMap, a U.S. Census Bureau application that provides maps showing where workers live and are

employed, has now been published with a variant of differential privacy.⁶

The following paper by Frank McSherry introduces a system called PINQ that integrates differential privacy into the C# LINQ framework, which adds database query functionality to C#. PINQ enables queries over data while elegantly hiding the complexity of the underlying differentially privacy mechanisms. Users of PINQ write programs that look almost identical to standard LINQ programs, but PINQ ensures that all query answers adhere to differential privacy, and it composes the information leakage from different queries until the privacy budget of the program has run out.

Differential privacy and PINQ give only a glimpse into a new exciting area at the confluence of ideas from computer science, statistics, law, and social sciences. I believe we will see much further progress on formal privacy definitions and improved methods, and I hope that future data products from the national statistics offices will be published with some formal notion of disclosure control.

Carrol Wright would be amazed by the field today. 

References

1. Agrawal, R. and Srikant, R. Privacy-preserving data mining. In *Proceedings of ACM SIGMOD* (May 2000). ACM Press, NY.
2. Dalenius, T. Towards a methodology for statistical disclosure control. *Statistik Tidskrift* 15 (1977), 429-444.
3. Dwork, C., McSherry, F., Nissim, K. and Smith, A. Calibrating noise to sensitivity in private data analysis. In *Proceedings of the 2006 TCC Conference* (Mar. 2006), Springer-Verlag, 265-284.
4. Goldberg, J.P. and Moye, W.T. The first hundred years of the Bureau of Labor Statistics. *Bureau of Labor Statistics*, 1985.
5. Lindell, Y. and Pinkas, B. Privacy preserving data mining. In *Proceedings of Crypto '00* (Aug. 2000), Springer-Verlag, 20-24.
6. U.S. Census Bureau's Longitudinal Employer-Household Dynamics Program OnTheMap Application; <http://lehdm4.did.census.gov>
7. Warner, S. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association* (1965), 63-69.

Johannes Gehrke (Johannes@cs.cornell.edu) is a professor in the Department of Computer Sciences at Cornell University, Ithaca, NY.

© 2010 ACM 0001-0782/10/0900 \$10.00

Privacy Integrated Queries:

An Extensible Platform for Privacy-Preserving Data Analysis

By Frank McSherry

Abstract

Privacy Integrated Queries (PINQ) is an extensible data analysis platform designed to provide unconditional privacy guarantees for the records of the underlying data sets. PINQ provides analysts with access to records through an SQL-like declarative language (LINQ) amidst otherwise arbitrary C# code. At the same time, the design of PINQ's analysis language and its careful implementation provide formal guarantees of differential privacy for any and all uses of the platform. PINQ's guarantees require no trust placed in the expertise or diligence of the analysts, broadening the scope for design and deployment of privacy-preserving data analyses, especially by privacy nonexperts.

1. INTRODUCTION

Vast quantities of individual information are currently collected and analyzed by a broad spectrum of organizations. While these data clearly hold great potential for analysis, they are commonly collected under the premise of privacy. Careless disclosures can cause harm to the data's subjects and jeopardize future access to such sensitive information.

This has led to substantial interest in data analysis techniques with guarantees of privacy for the underlying records. Despite significant progress in the design of such algorithms, privacy results are subtle, numerous, and largely disparate. Myriad definitions, assumptions, and guarantees challenge even privacy experts to assess and adapt new techniques. Careful and diligent collaborations between nonexpert data analysts and data providers are all but impossible.

In an attempt to put much of the successful privacy research in the hands of privacy nonexperts, we designed the Privacy Integrated Queries (PINQ) language and runtime, in which all analyses are guaranteed to have one of the strongest unconditional privacy guarantees: *differential privacy*.^{5, 8} Differential privacy requires that computations be formally indistinguishable when run with and without any one record, almost as if each participant had opted out of the data set. PINQ comprises a declarative programming language in which all written statements provide differential privacy, and an execution environment whose implementation respects the formal requirements of differential privacy.

Importantly, the privacy guarantees are provided by the platform itself; they require no privacy sophistication on the part of the platform's users. This is unlike much prior privacy research which often relies heavily on expert design and analysis to create analyses, and expert evaluation to vet proposed approaches. In such a mode, nonexpert analysts are unable to express themselves clearly or convincingly, and

nonexpert providers are unable to verify or interpret their privacy guarantees. Here the platform itself serves as a common basis for trust, even for analysts and providers with no privacy expertise.

The advantage our approach has over prior platforms lies in differential privacy: its robust guarantees are compatible with many declarative operations and permit end-to-end analysis of arbitrary programs containing these operations. Its guarantees hold in the presence of arbitrary prior knowledge and for arbitrary subsequent behavior, simplifying the attack model and allowing realistic, incremental deployment. Its formal nature also enables unexpected new functionality, including grouping and joining records on sensitive attributes, the analysis of text and unstructured binary data, modular algorithm design (i.e., without whole-program knowledge), and analyses which integrate multiple data sources from distinct and mutually distrustful data providers.

The main restriction of this approach is that analysts can only operate on the data from a distance: the operations are restricted to declarative transformations and aggregations; no source or derived records are returned to the analysts. This restriction is not entirely unfamiliar to many analysts, who are unable to personally inspect large volumes of data. Instead, they write computer programs to distill the data to manageable aggregates, on which they base further analyses. While the proposed platform introduces a stricter boundary between analyst and data, it is not an entirely new one.

1.1. An overview of PINQ

We start by sketching the different aspects of PINQ that come together to provide a data analysis platform with differential privacy guarantees. Each of these sections are then developed further in the remaining sections of the note, but the high level descriptions here should give a taste for the different facets of the project.

Mathematics of PINQ: The mathematical basis of PINQ, differential privacy, requires any outcome of a computation over a set of records be almost as likely with and without any one of those records. Computations with this guarantee behave, from the point of view of each participant, as if their data were never used. It is currently one of the strongest of privacy guarantees. The simplest example of a differentially private computation is noisy counting: releasing of the number of records in a data set perturbed by symmetric exponen-

The original version of this paper is entitled "Privacy Integrated Queries" and was published in the *Proceedings of SIGMOD 2009*.

tial (Laplace) noise. Many other simple aggregations (e.g., Sum, Average, Median, among others) have similarly accurate randomized analogs.

To allow nonexpert analysts to produce new differentially private computations, we introduce the use of transformations, applied to data before differentially private aggregations, without weakening the differential privacy guarantees. For several relational transformations, a changed record in the input data set always results in relatively few changes in the output data set. A differentially private analysis applied to transformed data masks changes in the transformation's output, and consequently masks changes in its input as well. The composed transformation and analysis will provide differential privacy, with a formal guarantee depending on the quantitative form of "relatively few," which we must determine for each transformation. Such transformations can be composed arbitrarily, by nonexpert analysts, and combined with differentially private aggregations will serve as our query language.

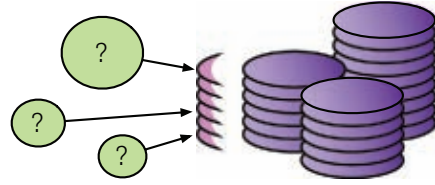
Finally, any sequence of differentially private computations also provides differential privacy; the quantitative privacy depletions are at worst additive (and occasionally better), and can be tracked online. Consequently, we can embed the query language above into any general purpose programming language, allowing arbitrary use of the results that return from the queries, as long as we monitor and constrain the total privacy depletion.

Implementation of PINQ: We have implemented a prototype of PINQ based on C#'s Language Integrated Queries (LINQ), an SQL-like declarative query language extension to .NET languages. Data providers use PINQ to wrap arbitrary LINQ data sources with a specified differential privacy allotment for each analyst. Analysts then write arbitrary C# programs, writing queries over PINQ data sources almost as if they were using unprotected LINQ data sources. PINQ's restricted language and run-time checks ensure that the provider's differential privacy requirements are respected, no matter how an analyst uses these protected data sets.

At a high level, PINQ allows the analyst to compose arbitrary queries over the source data, whose quantitative differential privacy guarantees are evaluated before the query is executed. If the analyst has framed a query whose privacy cost falls within the bounds prescribed by the data providers, the query is executed and the privacy cost subtracted from the amount available to the analyst for the associated data sets. If the cost falls outside the bounds, PINQ does not execute the query.

PINQ is designed as a thin layer in front of an existing query engine (Figure 1); it does not manage data or execute queries. Instead, it supplies differentially private implementations of common transformations and aggregations, themselves written in LINQ and executed by the LINQ providers of the underlying data sets. This approach substantially simplifies our implementation, but also allows a large degree of flexibility in its deployment. A data source only needs a LINQ interface to support PINQ, and we can take advantage of any investment and engineering put in to

Figure 1. PINQ provides a thin protective layer in front of existing data sources, presenting an interface that appears to be that of the raw data itself.



performant LINQ providers.

We stress that PINQ represents a very modest code base; in its current implementation it is only 613 lines of C# code. The assessment logic, following the math, is uncomplicated. The aggregations must be carefully implemented to provide differential privacy, but these are most often only a matter of postprocessing the correct aggregate (e.g., adding noise). PINQ must also ensure that the submitted queries conform to our mathematical model for them. LINQ achieves substantial power by allowing general C# computations in predicates of `Where`, functions of `Select`, and other operations. PINQ must restrict and shepherd these computations to mitigate the potential for exploitation of side channels.

Applications of PINQ: Programming with PINQ is done through the declarative LINQ language, in an otherwise unconstrained C# program. The analyst is not given direct access to the underlying data; instead, information is extracted via PINQ's aggregation methods. In exchange for this indirection, the analyst's code is allowed to operate on unmasked, unaltered, live records.

With a few important exceptions, programs written with PINQ look almost identical to their counterparts in LINQ. The analysts assemble an arbitrary query from permitted transformations, and specify the accuracy for aggregations. Example 1 contains a C# PINQ fragment for counting distinct IP addresses issuing searches for an input query phrase.

EXAMPLE 1. COUNTING SEARCHES FROM DISTINCT USERS IN PINQ.

We will develop this example into a more complex search log visualization application showcasing several of PINQ's

```
var data = new PINQueryable<SearchRecord>(... ..);

var users = from record in data
            where record.Query == argv[0]
            groupby record.IPAddress;

Console.WriteLine(argv[0] + ":" + users.Count(0.1));
```

advantages over other approaches: rich data types, complex transformations, and integration into higher level applications, among many others. The full application is under 100 lines of code and took less than a day to write.

We have written several other examples of data analyses in PINQ, including k-means clustering, perceptron

classification, and contingency table measurement. These examples have all been relatively easy adaptations of existing approaches.^{2,4}

2. MATHEMATICAL FOUNDATIONS

We now develop some supporting mathematics for PINQ. We review the privacy definition we use, differential privacy, and develop several properties necessary to design a programming language supporting its guarantees. Specifically, we discuss the data types we can support, common differentially private aggregations, how transformations of the data sets impact privacy, and how privacy guarantees of multiple analyses compose. All of our conclusions are immediate consequences of differential privacy, rather than additional assumptions or implementation details. The proofs are available in the full version of the paper.¹⁰

2.1. Differential privacy

Differential privacy is a relatively new privacy definition, building upon the work of Dwork et al.⁸ and publicly articulated in Dwork.⁵ It differs from most previous definitions in that it does not attempt to guarantee the prevention of data disclosures, privacy violations, or other bad events; instead, it guarantees that participation in the data set is not their cause.

The definition of differential privacy requires that a randomized computation yield nearly identical distributions over outcomes when executed on nearly identical input data sets. Treating the input data sets as multisets of records over an arbitrary domain and using $\ominus$ for symmetric difference (i.e., $A \ominus B$ is the set of records in A or B , but not both):

DEFINITION 1. A randomized computation M provides ϵ -differential privacy if for any two input data sets A and B , and any set of possible outputs S of M ,

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] \times \exp(\epsilon \times |A \ominus B|).$$

For values of x much less than one, $\exp(x)$ is approximately $1 + x$. Differential privacy relies only on the assumption that the data sets are comprised of records, of any data type, and is most meaningful when there are few records for each participant, relative to $1/\epsilon$.

The definition is not difficult to motivate to nonexperts. A potential participant can choose between two inputs to the computation M : a data set containing their records (A) and the equivalent data set with their records removed (B). Their privacy concerns stem from the belief that these two inputs may lead to noticeably different outcomes for them. However, differential privacy requires that *any* output event (S) is almost as likely to occur with these records as without. From the point of view of any participant, computations which provide differential privacy behave almost as if their records had not been included in the analysis.

Taking a concrete example, consider the sensible concern of most Web search users that their name and search history might appear on the front page of the *New York Times*.³ For each participant, there is some set S of outputs of M that would prompt the *New York Times* to this publication; we do not necessarily know what this set S of outputs is,

but we need not define S for the privacy guarantees to hold. For all users, differential privacy ensures that the probability the *New York Times* publishes their name and search history is barely more than had it not been included as input to M . Unless the user has made the queries public in some other way, we imagine that this is improbable indeed.

2.2. Basic aggregations

The simplest differentially private aggregation (from Dwork et al.⁸) releases the number of records in a data set, after the addition of symmetric exponential (Laplace) noise, scaled by ϵ (Figure 2). The Laplace distribution is chosen because it has the property that the probability of any outcome decrease by a factor of $\exp(\epsilon)$ with each unit step away from its mean. Translating its mean (shifting the true value) by one unit scales the probability of any output by a multiplicative factor of at most $\exp(\epsilon)$. Changing an input data set from A to B can shift the true count by at most $|A \ominus B|$, and consequently a multiplicative change of at most $\exp(\epsilon \times |A \ominus B|)$ in the probability of any outcome.

THEOREM 1. The mechanism $M(X) = |X| + \text{Laplace}(1/\epsilon)$ provides ϵ -differential privacy.

The Laplace distribution has exponential tails in both directions, and the probability that the error exceeds t/ϵ in either direction is exponentially small in t . Consequently, the released counts are likely to be close to the true counts.

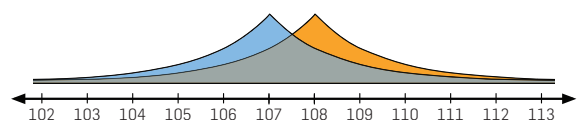
Other Primitive Aggregations: There are many other mechanisms that provide differential privacy; papers on the subject typically contain several. To date each has privacy established as above, by written mathematical proof based on intended behavior. While this is clearly an important step in developing such a computation, the guarantees are only as convincing as the proof is accessible and the implementation is correct.

Our goal is to enable the creation of as many differentially private computations as possible using only a few primitive components, whose mathematical properties and implementations can be publicly scrutinized and possibly verified. While we shouldn't preclude the introduction of novel primitives, they should be the exceptional, rather than default, approach to designing new differentially private algorithms.

2.3. Stable transformations

Rather than restrict programmers to a fixed set of aggregations, we intend to supply analysts with a programming language they can use to describe new and unforeseen computations. Most of the power of PINQ lies in arming the

Figure 2. Adding symmetric exponential noise to counts causes the probability of any output (or set of outputs) to increase or decrease by at most a multiplicative factor when the counts are translated.



analyst with a rich set of transformations to apply to the data set before differentially private aggregations.

DEFINITION 2. We say a transformation T is c -stable if for any two input data sets A and B ,

$$|T(A) \ominus T(B)| \leq c \times |A \ominus B|.$$

Transformations with bounded stability propagate differential privacy guarantees made of their outputs back to their inputs, scaled by their stability constant.

THEOREM 2. Let M provide ϵ -differential privacy, and let T be an arbitrary c -stable transformation. The composite computation $M \circ T$ provides $(\epsilon \times c)$ -differential privacy.

Once stability bounds are established for a set of transformations, a nonexpert analyst can combine any number of them as they see fit. Differential privacy bounds result from repeated application of Theorem 2, compounding the stability constants of the applied transformations with the ϵ value of the final aggregation.

Example Transformations: To give a sense for the types of stability bounds to expect, we consider a few representative transformations from LINQ.

The `Where` transformation takes an arbitrary predicate over records, and results in the subset of records satisfying the predicate. Any records in difference between A and B will result in at most those records in difference between their restrictions, resulting in a stability of one. Importantly, this is true independent of the supplied predicate; the predicate's logic can use arbitrarily sensitive information in the records and will still have stability one.

The `GroupBy` transformation takes a key selection function, mapping records to some key type, and results in a set of groups, one for each observed key, where each group contains the records mapped to the associated key value. For every record in difference between A and B , a group in the output can *change*. A change corresponds to symmetric difference two, not one; despite the apparent similarities in the groups, subsequent logic (e.g., a `Where`) can treat the two groups as arbitrarily different. As with `Where`, the stability of two holds for any key selection function, including those based on very sensitive fields or functions thereof.

The `Union` transformation takes a second data set, and results in the set of elements in either the first or the second data set. A record in difference between A and B results in no more than one record in difference in the output, yielding stability one. This is also true for records in difference in the second data set, giving us an example of a binary transformation. A differentially private analysis of the result of a binary transformation reflects information about both sources. This is uncomplicated unless the inputs derive from common data. Even so, a single change to a data set in common induces a bounded change in each of the transformation's inputs, and a bounded change in its output (i.e., the stability constants add).

The `Join` transformation takes a second data set, and key selection functions for both data sets. It intends to

result in a set of pairs of records, one from each input, of records whose keys match. A single record in either set could match an unbounded number of records in the other set. Consequently, this important transformation has no stability bound. As we discuss later, there are restricted forms of `Join` that do have bounded stability (stability one, with respect to both inputs), but their semantics deviate from the unrestricted `Join` present in LINQ.

2.4. Composition

Any sequence of differentially private computations also provides differential privacy. Importantly, this is true even when subsequent computations can depend arbitrarily on the outcomes of the preceding computations.

THEOREM 3. Let M_i each provide ϵ_i -differential privacy. The sequence of $M_i(X)$ provides $(\sum_i \epsilon_i)$ -differential privacy.

This simple theorem indicates that to track the cumulative privacy implications of several analyses, we need not do anything more complicated than add the privacy depletions.

If the queries are applied to disjoint subsets of the input domain we can improve the bound to the worst of the privacy guarantees, rather than the sum.

THEOREM 4. Let M_i each provide ϵ -differential privacy. Let D_i be arbitrary disjoint subsets of the input domain D . The sequence of $M_i(X \cap D_i)$ provides ϵ -differential privacy.

Whereas sequential composition is critical for any functional privacy platform, parallel composition is a very important part of extracting good performance from a privacy platform. Although such operations could be analyzed as sequential composition, the privacy guarantee would scale with the number of subsets analyzed, often quite large.

2.5. A privacy calculus

The mathematics of this section allows us to quantitatively bound the privacy implications of arbitrary sequences of rich transformations and aggregations. This simplicity allows us to avoid burdening the analyst with the responsibility of correctly or completely describing the mathematical features of their query. Even for researchers familiar with the mathematics (e.g., the author), the reasoning process can be quite subtle and error-prone. Fortunately, it can be automated, the subject of Section 3.

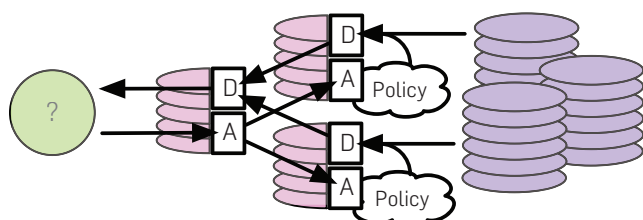
3. IMPLEMENTING PINQ

PINQ is built atop C#'s LINQ. LINQ is a recent language extension to the .NET framework integrating declarative access to data streams (using a language very much like SQL) into arbitrary C# programs. Central to LINQ is the `IQueryable<T>` type, a generic sequence of records of type `T`. An `IQueryable` admits transformations such as `Where`, `GroupBy`, `Union`, `Join`, and more, returning new `IQueryable` objects over possibly new types. Only once an aggregation or enumeration is

PINQ's implementation centers on a `PINQueryable<T>` generic type, wrapped around an underlying `IQueryable<T>`. This type supports the same methods as an `IQueryable`, but with implementations ensuring that the appropriate privacy calculations are conducted before any execution is invoked. Each `PINQueryable` is comprised of a private member data set (an `IQueryable`), and a second new data type, a `PINQAgent`, responsible for accepting or rejecting requested increments to epsilon. Aggregations test the associated `PINQAgent` to confirm that the increment to epsilon is acceptable before they execute. Transformations result in new `PINQueryable` objects with a transformed data source and a new `PINQAgent`, containing transformation-appropriate logic to forward modified epsilon requests to the agents of its source `PINQueryable` data sets.

3.1. Aggregation operators

Figure 3. PINQ control/data flow. An analyst initiates a request to a PINQ object, whose agent (A) confirms, recursively, differentially private access. Once approved by the providers' agents, data (D) flows back through trusted code ensuring the appropriate level of differential privacy.



EXAMPLE 2. [ABBREVIATED] IMPLEMENTATION OF COUNT.

```
double Count(double epsilon)
{
    if (epsilon > 0.0 && myagent.Alert(epsilon))
        return mysource.Count() + Laplace(1.0/epsilon);
    else
        throw new Exception("Access is denied");
}
```

The implementations of these methods and the proofs of their privacy guarantees are largely prior work. Sum, like Count, is implemented via the addition of Laplace noise and is discussed in Dwork et al.⁸ Average and Median are implemented using the exponential mechanism of McSherry and Talwar,¹¹ and output values in the range $[-1, +1]$ with probabilities

$$\Pr[\text{Median}(A)=x] \propto \max_{\text{med}(B)=x} \exp(-\epsilon \times |A \ominus B|/2)$$

$$\Pr[\text{Average}(A) = x] \propto \max_{\text{avg}(B)=x} \exp(-\epsilon \times |A \ominus B|/2)$$

The accuracy of *Average* is roughly $2/\epsilon$ divided by the number of records in the data set. *Median* results in a value which partitions the input records into two sets whose sizes differ by roughly an additive $2/\epsilon$; it need not be numerically close to the actual median.

PINQ’s flexibility derives from its transformation operators, each of which results in a new `PINQueryable` wrapped around an updated data source. The associated `PINQAgent` is wired to forward requests on to the participating source data sets before accepting, scaling `epsilon` by the transformation’s stability constant.

Our implementations of many transformations are mostly a matter of constructing new `PINQueryable` and `PINAgent` objects with the appropriate parameters. Some care is taken to restrict computations, as discussed in Section 3.4. Example 3 depicts the implementation of `PINQ`'s `GroupBy`. Most transformations require similarly simple privacy logic.

EXAMPLE 3. [ABBREVIATED] IMPLEMENTATION OF GROUPBY.

```

PINQueryable<IGrouping<K,T>>
GroupBy<T,K>(Expression<Func<T,K>> keyFunc)
{
    // Section 3.4 explains this, and why it is needed
    keyFunc = Purify(keyFunc) as Expression<Func<T,K>>;

    // new agent with appropriate ancestor and stability
    var newagent = new PINQAgentUnary(this.agent, 2.0);

    // new data source reflecting the operation
    var newsource = this.source.GroupBy(keyFunc);

    // construct and return a new source and agent pair
    return new PINQueryable<IGrouping<K,T>>(newsource,
                                           newagent);
}

```

The `Join` transformation is our main deviation from LINQ. To ensure stability one with respect to each input, we only report pairs that are the result of unique key matches. To ensure these semantics, PINQ's `Join` invokes LINQ's `GroupBy` on each input, using their key selection functions. Groups with more than one element are discarded, and the resulting singleton elements are joined using LINQ's `Join`.

While this clearly (and intentionally) interferes with standard uses of `Join`, analysts can reproduce its standard behavior by first invoking `GroupBy` on each data set, ensuring that there is at most one record per group, before invoking `Join`. The difference is that the `Join` is now required to reduce pairs of groups, rather than pairs of records. Each pair of groups yields a single result, rather than the unbounded cartesian product of the two, constraining the output but enabling privacy guarantees.

3.3. The partition operator

Theorem 4 tells us that structurally disjoint queries cost only the maximum privacy differential, and we would like to expose this functionality to the analyst. To that end, we introduce a `Partition` operation, like `GroupBy`, but in which the analyst must explicitly provide a set of candidate keys. The analyst is rewarded with a set of `PINQueryable` objects, one for each candidate key, containing the (possibly empty) subset of records that map to the each of the associated keys. It is important that PINQ does not reveal the set of keys present in the actual data, as this would violate differential privacy. For this reason, the analyst must specify the keys of interest, and PINQ must not correct them. Some subsets may be empty, and some records may not be reflected in any subset.

The `PINQAgent` objects of these new `PINQueryable` objects all reference the same source `PINQAgent`, of the source data, but following Theorem 4 will alert the agent only to changes in the maximum value of `epsilon`. The agents share a vector of their accumulated `epsilon` values since construction, and consult this vector with each update to see if the maximum has increased. If so, they forward the change in maximum. If the maximum has not increased, they accept the request.

The difference between the uses of `GroupBy` and

`Partition` in PINQ can be seen in the following two queries:

- Q1. How many ZIP codes contain at least 10 patients?
- Q2. For each ZIP code, how many patients live there?

For Q1, a `GroupBy` by ZIP, a `Where` on the number of patients, and a `Count` gives an approximate answer to the exact number of ZIP codes with at least 10 patients. For Q2, a `Partition` by ZIP, followed by a `Count` on each part returns an approximate count for each ZIP code. As the measurements can be noisy, neither query necessarily provides a good estimate for the other. However, both are at times important questions, and PINQ is able to answer either accurately depending on how the question is posed.

The `Partition` operator can be followed not only by aggregation but by further differentially private computation on each of the parts. It enables a powerful recursive descent programming paradigm demonstrated in Section 4, and is very important in most nontrivial data analyses.

3.4. Security issues in implementation

Although the stability mathematics, composition properties, and definition of differential privacy provide mathematical guarantees, they do so only when PINQ's behavior is in line with our mathematical expectations. There are many important but subtle implementation details intended to protect against clever attackers who might use the implementation details of PINQ to learn information the mathematics would conceal. These are largely the result of user-defined code that may attempt to pass information out through side channels, either directly through disk or network channels, or indirectly by throwing exceptions or simply not terminating. PINQ's `purify` function gives the provider the opportunity to examine incoming methods and rewrite them, either by restricting the computations to those comprised of known-safe methods, or by rewriting the methods with appropriate guards. There are other issues and countermeasures in the full paper, and likely more unrecognized issues to be discovered and addressed.

4. APPLICATIONS AND EVALUATION

In this section, we present data analyses written with PINQ. Clearly not all analysis tasks can be implemented in PINQ (indeed, this is the point), but we aim to convince the reader that the set is sufficiently large as to be broadly useful.

Our main example application is a data visualization based on Web search logs containing IP addresses and query text. The application demonstrates many features of PINQ largely absent from other privacy-preserving data analysis platforms. These include direct access to unmodified data, user-supplied record-to-record transformations, operations such as `GroupBy` and `Join` on "sensitive" attributes, multiple independent data sets, and unfettered integration into higher-level programs.

For our experiments we use the DryadLINQ¹⁵ provider. DryadLINQ is a research LINQ provider implemented on top of the Dryad⁹ middleware for data parallel computation,

and currently scales to at least thousands of compute nodes. Our test data sets are of limited size, roughly 100GB, and do not fully exercise the scalability of the DryadLINQ provider. We do not report on execution times, as PINQ's reasoning is an insignificant contribution, but rather the amount and nature of information we can extract from the data privately.

For clarity, we present examples written as if the data analyst is also the data provider, charged with assembling the source PINQueryable objects. In a real deployment, this assembly should be done on separate trusted infrastructure.

4.1. Data analysis: Stage 1 of 3

We start with a simple application of PINQ, approximating the number of distinct search users who have searched for an arbitrary query term. Our approach is just as it would have been in LINQ: we first transform the search records (comma-delimited strings) into tuples (string arrays) whose fields have known meaning, then restrict the data to records with the input search query, then group by the supplied IP address to get a proxy for distinct users, then count the remaining records (groups of string arrays). The program is reproduced in Example 4.

EXAMPLE 4. MEASURING QUERY FREQUENCIES IN PINQ.

```
// prepare data with privacy budget
var agent = new PINQAgentBudget(1.0);
var data = new PINQueryable<string>(rawdata, agent);

// break out fields, filter by query, group by IP
var users = data.Select(line => line.Split(','))
    .Where(fields => fields[20] == args[0])
    .GroupBy(fields => fields[0]);

// output the count to the screen, or anywhere else
Console.WriteLine(args[0] + ":" + users.Count(0.1));
```

This relatively simple example demonstrates several important features of PINQ. The input data are text strings; we happen to know a priori that they are comma delimited, but this information plays no role in the privacy guarantees. The filtering is done against an analyst-supplied query term, and may be frequent or infrequent, sensitive or insensitive. To get the set of distinct users we group using the logged IP address, clearly highly sensitive information. Despite these uncertainties about the analysis, the differential privacy guarantees are immediately quantifiable.

4.2. Data analysis: Stage 2 of 3

Our program as written gives the count for a single query, and if the analyst wants additional counts they must run the program again. This incurs additional privacy cost, and will be unsuitable for extracting large numbers of query counts.

Instead, we can rewrite the previous program to use the Partition operator to permit an arbitrary number of counts at fixed privacy cost. Rather than filter records with Where, we use the same key selection function and an input set of query strings to Partition the records. Having done so, we iterate through each of the queries

and associated parts, grouping the records in each by IP address. To further enrich the example, we then partition each of these data sets by the number of times each IP address has issued the query, before producing a noisy count (see Example 5).

EXAMPLE 5. MEASURING MANY QUERY FREQUENCIES IN PINQ.

```
// prepare data with privacy budget
var agent = new PINQAgentBudget(1.0);
var data = new PINQueryable<string>(rawdata, agent);

// break out fields, but partition rather than filter
var parts = data.Select(line => line.Split(','))
    .Partition(args, fields =>
        fields[20]);

foreach (var query in args)
{
    // use the searches for query, grouped by IP address
    var users = parts[query].GroupBy(fields => fields[0]);

    // further partition by the frequency of searches
    var freqs = users.Partition(new int[] { 1,2,3,4,5 },
        group => group.Count());

    // output the counts to the screen, or anywhere else
    Console.WriteLine(query + ":");
    foreach (var count in new int[] { 1,2,3,4,5 })
        Console.WriteLine(freqs[count].Count(0.1));
}
```

Because we use Partition rather than multiple Where calls, the privacy cost associated with the program can be seen by PINQ to be only the maximum of the privacy costs of each of the loops, exactly the same cost as in Example 4.

Table 1 reports the measurements of a few query strings taken over our data set. Each reported measurement is the exact count plus Laplace noise with parameter 10, corresponding to standard deviation $10\sqrt{2}$. For most measurements this error is relatively insignificant. For some measurements it is significant, but nonetheless reveals that the original value is quite small.

4.3. Data analysis: Stage 3 of 3

We now expand out our example program from simple

Table 1. Numbers of Users Searching for Various Terms, Broken Out by Number of Times They Searched.

	Freq 1	Freq 2	Freq 3	Freq 4	Freq 5
GOOGLE	356,743	108,336	45,363	25,092	14,347
YAHOO	140,966	42,379	17,624	9671	5,707
BAIDU	300	79	29	26	9
AMAZON	16,798	3,376	808	378	132
EBAY	100,338	26,205	9,564	4,065	2,604
CNN	25,442	7,492	2,899	1,658	919
MSNBC	7,828	2,496	849	565	283

reporting (a not uncommon task) to a richer analysis application. Our goal is to visualize the distribution of locations of searches for various search queries. At a high level, we will transform the IP addresses into latitude-longitude pairs, by joining with a second proprietary data set, and then send the coordinates to a visualization algorithm borrowed from the work of McSherry and Talwar.¹² Although we will describe the visualization algorithm at a high level, it is fundamental that PINQ provides privacy guarantees even without the knowledge of what the algorithm plans to do with the data.

Starting from the prior examples, in which we have partitioned the data sets by query and grouped the results by IP address, we now demonstrate a fragment that will let us transform IP addresses into latitude-longitude coordinates. We use a second data set `iplatlon` whose entries are IP addresses and corresponding latitude-longitude coordinates. We join these two data sets, using the IP addresses in each as keys, resulting in a lat-lon coordinate pair in place of each group of searches. Example 6 contains the code for this `Join` transformation.

EXAMPLE 6. TRANSFORMING IP ADDRESSES TO COORDINATES.

```
// ... within the per-query loop, from before ...

// use the searches for query, group by IP address
var users = parts[query].GroupBy(fields => fields[0]);

// extract IP address from each group, and match
var coords = users.Join(iplatlon,
    group => group.Key,
    entry => entry.IP,
    (group, entry) => entry.LatLon);
```

Recall that `Join` in PINQ only reports pairs that result from unique matches. In this program, we know that each IP occurs as a key at most once in `users`, as we have just performed a `GroupBy` with this field as the key. In the second data set, we assume (perhaps wrongly) that there is one entry per IP address. If this is not the case, or if we are not sure, we could also group the second data set by IP address and use the first lat-lon entry in each group. This is arguably more robust, but results in an additional factor of two in the privacy cost against the `iplatlon` data set; we would like to avoid this cost when we know we can safely do so.

Finally, our algorithm takes the list of lat-lon coordinates of the IPs searching for the input search query, and invokes a Visualization subroutine, whose implementation is not specified here. An example for the query “cricket” can be seen in Figure 4. Readers who are not entirely sure how or why this routine works are in roughly the same situation as most data providers. We have no intuition as to why the computation should be preserve privacy, nor is any forthcoming. Nonetheless, as the routine is only provided access to the data through a `PINQueryable`, we are assured of differential privacy guarantees even without understanding the algorithm’s intent or implementation.

Support for such “modular design” of privacy algorithms is an important enabler for research and development, removing the need for end-to-end understanding of the computation. This is especially important for exploratory data analysis, where even the analysts themselves may not know the questions they will need answered until they start asking them. Removing the requirement of whole-program understanding also enables proprietary data analyses, in which an analyst may not want to divulge the analysis they intend to conduct. While the execution platform clearly must be instructed in the computations the analyst requires, the data provider does not need to be informed of their specifics to have privacy assurances.

Our second data set raises an interesting point about alternate applications of differential privacy. While the operation we perform, mapping IP addresses to latitude-longitude pairs, is essentially just a complicated `Select`, the data set describing the mapping is proprietary. Each record in the data set required some investment of effort to produce, from which the owners presumably hope to extract value; they may not want to release the records in the clear. Using this data set through PINQ prevents the dissemination of individual records, preserving the value of the data set while still permitting its use in analyses. Similarly, many organizations have data retention policies requiring the deletion of data after a certain amount of time. Ensuring that this deletion happens when analysts are allowed to create their own copies of the data is effectively impossible. Again, PINQ allows analysts to use such data without compromising the organization’s obligations.

Figure 4. Example output, displaying a representative distribution of the latitude-longitude coordinates of users searching for “cricket.” The computation has differential privacy not because of properties of the output itself, a quite complicated artifact, but because of the manner in which it was produced.



5. RELATED WORK

The analysis of sensitive data under the constraints of confidentiality has been the subject of a substantial amount of prior research; for an introductory survey we recommend the reader to Adam and Wortmann,¹ but stress that the field is still very much evolving. For an introduction to differential privacy we recommend the reader to Dwork.⁶

While PINQ is the first platform we are aware of providing differential privacy guarantees, several other interactive data analysis platforms have been proposed as an approach to providing privacy guarantees. Such platforms are generally built on the principle that aggregate values are less sensitive than individual records, but are very aware that allowing an analyst to define an arbitrary aggregation is very dangerous. Various and varying criteria are used to determine which aggregates an analyst should be able to conduct. To the best of our knowledge, none of these systems have provided quantifiable end-to-end privacy guarantees.

Recent interest in differential privacy for interactive systems appears to have started with Mirkovic,¹³ who proposed using differential privacy as a criteria for admitting analyst-defined aggregations. The work defines an analysis language (targeted at network trace analysis) but does not go so far as to specify semantics that provide formal differential privacy guarantees. It seems possible that PINQ could support much of the proposed language without much additional work, with further trace-specific transformations and aggregations added as extensions to PINQ.

Airavat¹⁴ is a recent analogue of PINQ for Map-Reduce computations. The authors invest much more effort in hardening the system, securing the computation through the use of a mandatory access control operating system and an instrumented java virtual machine, as well as PINQ-style differential privacy mathematics. At the same time, it seems that the resulting analysis language (one Map-Reduce stage) is less expressive than LINQ. It remains to be seen to what degree the system level guarantees of Airavat can be fruitfully hybridized with the language level restriction used in PINQ.

6. CONCLUSION

We have presented “Privacy Integrated Queries” (PINQ), a trustworthy platform for privacy-preserving data analysis. PINQ provides private access to arbitrarily sensitive data, without requiring privacy expertise of analysts or providers. The interface and behavior are very much like that of Language Integrated Queries (LINQ), and the privacy guarantees are the unconditional guarantees of differential privacy.

PINQ presents an opportunity to establish a more formal and transparent basis for privacy technology and research. PINQ’s contribution is not only that one can write private programs, but that one can write only private programs. Algorithms built out of trusted components inherit privacy properties structurally, and do not require expert analysis and understanding to safely deploy. This expands the set of capable users of sensitive data, increases the portability

of privacy-preserving algorithms across data sets and domains, and broadens the scope of the analysis of sensitive data.

6.1. Availability

The prototype of PINQ used for the experiments in this paper, as well as further example programs and a brief tutorial, are available at <http://research.microsoft.com/PINQ>.

Acknowledgments

The author gratefully acknowledges the contributions of several collaborators. Ilya Mironov, Kobbi Nissim, and Adam Smith have each expressed substantial interest in and support for privacy tools and technology usable by nonexperts. Yuan Yu, Dennis Fetterly, Úlfar Erlingsson, and Mihai Budiu helped tremendously in educating the author about LINQ, and have informed the design and implementation of PINQ. Many readers and reviewers have provided comments that have substantially improved the presentation of this paper. 

References

1. Adam, N.R., Wortmann, J.C. Security-control methods for statistical databases: A comparative study, *ACM Comput. Surv.*, 21, 4 (1989), 515–556.
2. Barak, B., Chaudhuri, K., Dwork, C., Kale, S., McSherry, F., Talwar, K. Privacy, accuracy, and consistency too: a holistic solution to contingency table release, in *PODS (2007)*, 273–282.
3. Barbaro, M., Zeller Jr., T. A face is exposed for AOL searcher no. 4417749, *The New York Times*, August 9, 2006.
4. Blum, A., Dwork, C., McSherry, F., Nissim, K. Practical privacy: The SuLQ framework, in *PODS (2005)*, 128–138.
5. Dwork, C. Differential privacy, in *ICALP (2006)*, 1–12.
6. Dwork, C. A firm foundation for private data analysis, *Communications of the ACM, Association for Computing Machinery, Inc.*, 2010.
7. Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., Naor, M., Our data, ourselves: Privacy via distributed noise generation, in *EUROCRYPT (2006)*, 486–503.
8. Dwork, C., McSherry, F., Nissim, K., Smith, A. Calibrating noise to sensitivity in private data analysis, in *TCC (2006)*, 265–284.
9. Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D. Dryad: distributed data-parallel programs from sequential building blocks, in *EuroSys. ACM (2007)*, 59–72.
10. McSherry, F. Privacy integrated queries: an extensible platform for privacy-preserving data analysis, in *SIGMOD Conference (2009)*, 19–30.
11. McSherry, F., Talwar, K. Mechanism design via differential privacy, in *FOCS (2007)*, 94–103.
12. McSherry, F., Talwar, K. Synthetic data via differential privacy, Manuscript.
13. Mirkovic, J. Privacy-safe network trace sharing via secure queries, in *NDA (2008)*.
14. Roy, I., Setty, S.T., Kilzer, A., Shmatikov, V., Witchel, E. Airavat: Security and privacy for mapreduce, in *NSDI Conference (2010)*.
15. Yu, Y., Isard, M., Fetterly, D., Budiu, M., Erlingsson, U., Gunda, P.K., Currey, J. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language, in *OSDI (2008)*.

Frank McSherry (mcsherry@microsoft.com), Microsoft Research, SVC, Mountain View, CA.

Technical Perspective

Constraint Satisfaction Problems and Computational Complexity

By Mark Jerrum

CONSIDER THE FOLLOWING very general setting for computational problems. An instance is presented as a set of variables taking values in a finite domain, together with a set of constraints on those variables. Each constraint applies to a certain tuple of variables and imposes a certain relation on the values they may take. The imposed relations come from an agreed “constraint language.” We ask the question: Does there exist an assignment to the variables that simultaneously satisfies all the constraints? So, if the domain has nine elements and the constraint language contains the binary relation “not equal to,” then we could express the problem of whether the vertices of a given graph can be properly colored with just nine colors. Or, more picturesquely, we could specify an arbitrary instance of Sudoku. (Technically, we would also require nine unary relations picking out the nine values in the domain.)

It takes only a little imagination to come up with a wealth of problems in scheduling and planning that can be expressed as *Constraint Satisfaction Problems* (CSPs) within an appropriate constraint language. It should come as no surprise, then, that CSPs have an important place in the fields of artificial intelligence and operations research. But why should there be such an interest in CSPs within the computational complexity community? Well, each constraint language (set of allowed constraint relations) defines a particular class of CSPs, and, as the constraint language becomes richer, the computational complexity of the corresponding class of CSPs becomes potentially greater. Classifying the computational complexity of the infinite array of possible constraint languages provides a challenging and stimulating task for complexity theorists, and hopefully one that has some practical value too.

The starting point was a result of Schaefer, which completely classified the computational complexity of CSPs in the case of a two-element (Boolean) domain. He identified certain polynomial-time solvable cases, for example, where all constraints can be expressed as conjunctions of Horn clauses. But the remarkable part of his result is that all other cases are *NP*-complete. In other words, he identified a dichotomy within Boolean CSPs—under the assumption $P \neq NP$ —between ones solvable in polynomial time, and ones that are *NP*-complete. This is significant, as a similar dichotomy is known not to exist in *NP* itself. There is no contradiction here, as not every problem in *NP* is expressible as a CSP. For example, it is not possible to express the Hamiltonian Cycle problem (a stripped-down version of the infamous Traveling Salesman problem) in any finite constraint language.

An article by Feder and Vardi provided a major spur to work on the computational complexity of CSPs. It would not be appropriate to describe their work in detail here, beyond noting they conjectured that a complexity dichotomy holds for the entire class of CSPs, not just ones over the Boolean domain. It transpires that the technical difficulties encountered when attempting to generalize Schaefer’s dichotomy to domains with more than two values are extreme, and the Feder-Vardi conjecture remains unresolved to this day. Fortunately, much progress has been made on related problems and special cases, making this a lively and dynamic area.

In the following paper, Bulatov and Marx deal with just such a related problem. Suppose the domain is Boolean, and the constraint language consists of the single binary relation “nand.” This class of CSPs contains the *n*-Queens Problem since we can use this binary relation to express the rule that we

are not allowed to have a queen at this position and that one. Well, not quite, since we must also insist there are *n* queens in all on the board, to deny the trivial solution with no queens! One way to incorporate such problems into the CSP framework is by adding a global cardinality constraint that determines the number of variables that take on each of the values in the underlying domain, in this instance that *n* of the variables should be set to “true.” This extended class of “CCSPs” is the subject of the paper. Again, it is possible to imagine less frivolous examples of CCSPs than the 8-Queens Problem. For example, in determining locations for a number of facilities, one would want to ensure not only that various proximity and capacity constraints are met, but that the total number of facilities is within budget. Note that a cardinality constraint may increase computational complexity: the single relation “nand” in isolation leads to trivial CSPs (just set every variable to “false”), but adding a cardinality constraint yields the *NP*-complete problem “Maximum Independent Set.”

Aside from establishing a dichotomy theorem for CCSPs, the authors provide an excellent introduction to the concerns and techniques of researchers in the computational complexity of CSPs. Intuition having been quickly developed in a sequence of examples and counterexamples, the reader is introduced to some of the tricks of the trade. A basic challenge in analyzing the computational complexity of CSPs is the following: Although the constraint language may be small, nevertheless these explicitly given relations can be combined (together with some extra variables) to generate a rich variety of derived, or “pp-definable” relations. One effective way to gain control over the ramifications of pp-definable relations is to transfer to a dual world of so-called polymorphisms: operations under which the class of definable relations is invariant. All this rich circle of ideas may be glimpsed in this study of CSPs with cardinality constraints. 

Mark Jerrum (mrj@inf.ed.ac.uk) is a professor of pure mathematics at Queen Mary, University of London.

© 2010 ACM 0001-0782/10/0900 \$10.00

Constraint Satisfaction Problems and Global Cardinality Constraints

By Andrei A. Bulatov and Dániel Marx

Abstract

In a constraint satisfaction problem (CSP) the goal is to find an assignment of a given set of variables subject to specified constraints. A global cardinality constraint is an additional requirement that prescribes how many variables must be assigned a certain value. We study the complexity of the problem CCSP(Γ), the CSP with global cardinality constraints that allows only relations from the set Γ . The main result of this paper characterizes sets Γ that give rise to problems solvable in polynomial time, and states that the remaining such problems are NP-complete.

1. CONSTRAINT PROBLEMS

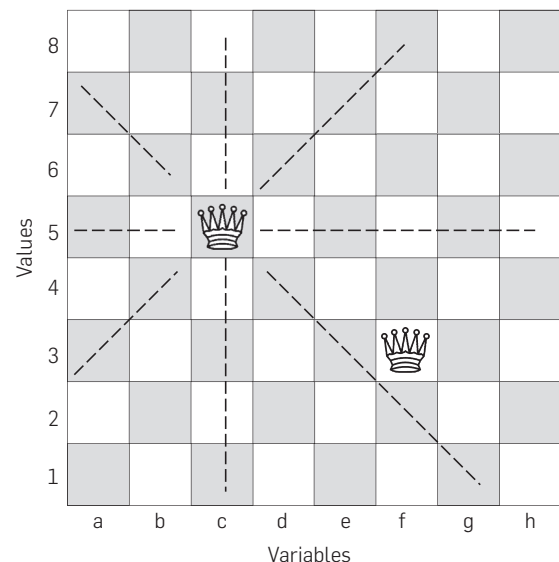
1.1. Constraint satisfaction problem

Among formalisms unifying and classifying various combinatorial problems the *Constraint Satisfaction Problem* (or CSP) is one of the most successful ones. In this problem, we are given a set of variables and a collection of restrictions—constraints—on the allowed combinations of values of the variables; the goal is to find an assignment to the variables so that all constraints are satisfied. Usually constraints are imposed on small sets of variables; thus, the CSP formalizes the idea of finding a global solution bound by local restrictions. The Sudoku puzzle gives a popular toy example of CSP. We need to assign values—numbers from 1 to 9—to variables—entries of the puzzle so that the values of variables in a row, column, or 3×3 block are different. Another toy example whose CSP encoding is less obvious is the 8-Queen problem: place eight queens on a 8×8 chessboard so that they do not hit each other.¹⁵ To represent it as a CSP we consider the columns $\{a, b, c, d, e, f, g, h\}$ (see Figure 1) as variables that can be assigned values from the set of rows, and the assigned value shows the position of a queen in this column.

Many combinatorial problems readily fall into this framework. For example, in the Graph 3-Coloring problem, the vertices of a given graph are variables to receive one of the three colors, and assignments are constrained by the requirement that adjacent vertices receive different colors. Thus, this problem is a CSP. The list of examples can be extended by other combinatorial problems like Satisfiability, problems in scheduling, temporal and spatial reasoning, and many others.

CSPs have been studied from both practical and theoretical perspectives. On the practical side, the expressive power of the CSP allows to model a wide range of real-world problems from planning²⁴ and scheduling,³⁵ frequency assignment problems,¹⁷ to image processing,³² to programming language analysis,³³ to natural language understanding.¹ A number of commercial and freeware solvers exist capable

Figure 1. The 8-Queen problem.



of solving a wide range of CSPs of nearly industrial scale, and methods of solving constraint problems are developing rapidly.¹⁵ On the theoretical side, researchers focus on several directions such as the complexity of CSPs problems, efficient algorithms for CSPs, where such algorithms exist, and connections of CSPs with other combinatorial problems.^{3, 8, 10, 13, 18, 21, 22, 26, 31, 34}

1.2. Global constraints

The ‘pure’ CSP described above is sometimes not enough to model practical problems, as some constraints that have to be satisfied are not ‘local’ in the sense that they cannot be viewed as applied to only a limited number of variables. Constraints of this type are called *global*. Global constraints are very diverse; the current Global Constraint Catalog (see <http://www.emn.fr/x-info/sdemasse/gccat/>) lists 313 types of such constraints. In this paper we focus on *global cardinality constraints*.^{6, 14}

Some of the global constraints such as the surjectivity of a solution, that is, the requirement that all variables take distinct values (cf. the Sudoku puzzle), allow simulation by local

The original version of this paper was published in the *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science* (Los Angeles, CA, Aug. 11–14, 2009), 419–428.

constraints. Surjectivity can be enforced by requiring that every two variables receive distinct values. However, sometimes it is not possible. In this paper we focus on one type of such ‘truly’ global constraints, *cardinality constraints*, that impose restrictions on the number of variables assigned certain values, see Figure 2. For instance, in the 3-Coloring problem, a cardinality constraint may require that at least half of the vertices of the graph are colored red.

1.3. Complexity of constraints

As the general CSP is NP-hard, the study of its complexity focuses on considering restricted versions of the problem. There are two principal ways to restrict the CSP, both of them can be applied to CSPs with cardinality constraints as well.

The first approach restricts the way constraints interact. The interaction of constraints can be represented by the *primal graph* whose vertices are variables, and two vertices are connected if and only if they belong to the scope of a constraint. This approach was motivated by the observation that if the primal graph is acyclic or close to acyclic in a well-defined sense (has bounded treewidth), then CSP becomes polynomial-time solvable.²⁰ Interestingly, attempts to characterize conjunctive queries to databases that can be processed efficiently led to the same question.²⁶ After a series of recent breakthrough results^{21, 31} the structure of polynomial-time solvable CSPs of this type is largely understood.

The second approach to restrict the CSP is to limit the allowed types of constraints. It can be expressed formally as follows. Let the possible values of variables in the problem be taken from a set D (the *domain*). In this paper, we always assume D to be finite. Then every constraint that can be imposed on a set of k variables is a list of all allowed combinations of values these variables can take simultaneously, that is, a k -ary relation on D . If now we fix a set Γ of such relations on D and allow constraints to be chosen only from Γ , we arrive to the problem denoted $\text{CSP}(\Gamma)$. In this context, Γ

is often called a *constraint language*. Same restrictions can be applied to problems with cardinality constraints. We use $\text{CCSP}(\Gamma)$ to denote such problem.

Problems of the form $\text{CSP}(\Gamma)$ and $\text{CCSP}(\Gamma)$ span a wide range of combinatorial problems such as ones in Figure 3, and many others.

GRAPH 3-COLORING. Let $R_{\neq_3}$ denote the disequality relation on a 3-element set, that is, the binary relation containing all pairs (a, b) of elements from the set such that $a \neq b$:

$$R_{\neq_3} = \begin{pmatrix} 1 & 1 & 2 & 2 & 3 & 3 \\ 2 & 3 & 1 & 3 & 1 & 2 \end{pmatrix}.$$

(Observe that we write pairs, and later longer tuples of elements vertically, so members of the relation are the columns of the matrix.) Then the 3-Coloring problem equals $\text{CSP}(\Gamma_{3\text{-Col}})$ where $\Gamma_{3\text{-Col}} = \{R_{\neq_3}\}$.

2-SATISFIABILITY. Recall that a literal is a propositional variable or its negation. A disjunction of literals (of 2 literals) is called a clause (a 2-clause). A propositional formula that is a conjunction of clauses (2-clauses) is said to be a conjunctive normal form, or a CNF (2-CNF) for short. In the 2-Satisfiability problem, given a 2-CNF, the goal is to find an assignment to its variables that makes the formula true. If the set of variables of the CNF is V then every clause defines a constraint on a pair of variables that forbids exactly one combination of values. Let $\Gamma_{2\text{-SAT}}$ be the following set of 4 binary relations, each of which omits a certain pair:

$$\begin{aligned} R_{x \vee y} &= \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} & R_{\bar{x} \vee y} &= \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \\ R_{x \vee \bar{y}} &= \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} & R_{\bar{x} \vee \bar{y}} &= \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

Then $\text{CSP}(\Gamma_{2\text{-SAT}})$ represents 2-Satisfiability and it is known to be polynomial-time solvable.

3-SATISFIABILITY. Analogously to 2-SAT, let $\Gamma_{3\text{-SAT}}$ be the set consisting of eight ternary relations on $\{0, 1\}$, each of which omits a certain triple. Then $\text{CSP}(\Gamma_{3\text{-SAT}})$ represents 3-Satisfiability and it is NP-complete.

INDEPENDENT SET. An independent set in a graph is a set of vertices, no two of which are connected with an edge. In the Independent Set problem, given a graph and a natural number k , the question is whether or not there exists an independent set of size k . Let

$$R_{\text{IS}} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix},$$

that is, R_{IS} excludes only $(1, 1)$, and $\Gamma_{\text{IS}} = \{R_{\text{IS}}\}$. Now, to reduce Independent Set to the CSP the vertices of a given graph are treated as variables and the constraint R_{IS} is imposed on every pair of adjacent vertices. For any solution of such CSP the variables (vertices) assigned 1 form an independent set in the graph. To express the restriction on the size of an independent set we can use a cardinality constraint that requires that exactly k variables

Figure 2. Formal definition of CSP and CCSP.

CSP

Let D be a (finite) set (the *domain*). Every instance $I = (V, \mathcal{C})$ of the problem CSP consists of:

- a set V of variables, and
- a set $\mathcal{C}$ of *constraints*. Every constraint is a pair $\langle \mathbf{s}, R \rangle$, where
 - $\mathbf{s} = (v_1, \dots, v_k)$ is a tuple of variables from V , not necessarily distinct, and
 - R is a k -ary relation over D .

A *solution* of $I = (V, \mathcal{C})$ is a mapping $\varphi: V \rightarrow D$ such that for any constraint $\langle \mathbf{s}, R \rangle$, we have $\varphi(\mathbf{s}) \in R$.

CCSP

A global cardinality constraint for an instance $I = (V, \mathcal{C})$ is a mapping $\pi: D \rightarrow \mathbb{N}$ such that $\sum_{a \in D} \pi(a) = |V|$. Solution φ satisfies π if $|\varphi^{-1}(a)| = \pi(a)$ for every $a \in D$. The question is whether or not there is a solution satisfying one of the given cardinality constraints.

$\text{CSP}(\Gamma)$ and $\text{CCSP}(\Gamma)$

Let Γ be a set (finite or infinite) of relations on D , called a *constraint language*. The problems $\text{CSP}(\Gamma)$ and $\text{CCSP}(\Gamma)$ include those instances of CSP and CCSP, respectively, that use only relations from Γ .

are assigned 1. Therefore Independent Set is equivalent to CCSP(Γ_{IS}). The Independent Set problem is well known to be NP-complete.

Despite such expressive power, problems of the form CSP(Γ) probably cannot capture all combinatorial problems. As is easily seen, all CSPs belong to the class NP. Some of them, such as 3-Coloring or 3-SAT are NP-complete, while others, for example, 2-SAT, belong to the class P, that is, solvable in polynomial time. If $P \neq NP$, there is an infinite hierarchy of complexity classes between P and NP such that problems from different classes are not reducible to each other in a natural sense.²⁸ However, all known problems CSP(Γ) turn out to be either in P or NP-complete. This phenomenon is known as complexity dichotomy.¹⁸ The dichotomy phenomenon was first discovered by Schaefer³⁴ for CSPs with 2-element domain, and was later confirmed in many particular cases.^{3, 7, 9} This caused Feder and Vardi to pose a conjecture, called the *Dichotomy Conjecture*, that every problem CSP(Γ) is either solvable in polynomial time or is NP-complete. The Dichotomy Conjecture remains open till now.

Remarkably, the phenomenon of complexity

dichotomy extends inside P, although a weaker notion of reduction is needed for this. To date, only four complexity classes and a series of very similar classes inside P are known such that CSP(Γ) can be complete in.^{2, 29} In some cases the lack of problems CSP(Γ) of intermediate complexity is shown.²⁹

In this paper, we report on a dichotomy theorem for CSPs with cardinality constraints. The next section describes a dynamic programming algorithm that solves CCSPs whenever it can be solved efficiently. In Section 3, we outline the algebraic approach to the CSP and CCSP and show how it can be used to formulate the dichotomy theorem for the CCSP. Finally, in Section 4 we present the main ideas behind the hardness result. A longer version of the paper can be found in.¹²

2. EASY CASES OF CCSP

2.1. Boolean CCSP

To gain some intuition we start with the Boolean CSP and CCSP, in which values are taken from the set $\{0, 1\}$. The dichotomy result for Boolean CSPs³⁴ identifies six types of tractable relations, that is, those which give rise to a CSP solvable in polynomial time. Among these relations are those representable by a 2-CNF, solution spaces of systems of linear equations over the 2-element field, and some others. If a constraint language Γ is not composed from relations of one of these six types, CSP(Γ) is NP-complete. For CCSPs, a dichotomy result was proved in Creignou et al.¹⁴ The structure of tractable CCSPs is much simpler. Let $R_{=2}$ and $R_{\neq 2}$ denote the equality and disequality relations on $\{0, 1\}$. Then CCSP(Γ) is solvable in polynomial time if and only if every relation from Γ can be expressed by a conjunction of $R_{=2}$ and $R_{\neq 2}$ clauses, and the two constant constraints 0 and 1. Otherwise the Bipartite Independent Set or Linear Equations problems can be reduced to CCSP(Γ), and the problem is NP-complete.

The polynomial-time solvable cases can be handled by a standard application of dynamic programming. Suppose that the instance is given by a set of binary equality/disequality clauses (see Figure 4 for a concrete example). Consider the graph formed by the binary clauses. There are at most two possible assignments for each connected component of the graph: setting the value of a variable uniquely determines the values of all the other variables in the component. Thus the problem is to select one of the two assignments for each component. Trying all possibilities would be exponential in the number of components. Instead, for $i = 1, 2, \dots$, we compute the set Π_i of all possible pairs (x, y) such that there is a partial solution on the first i components containing exactly x zeros and exactly y ones. It is not difficult to see that Π_{i+1} can be efficiently computed if Π_i is already known.

2.2. Generalizations

We generalize the results of Creignou et al.¹⁴ for arbitrary finite sets and arbitrary constraint languages. As usual, the characterization for arbitrary finite domains is significantly more complex and technical than for the 2-element domain. As a straightforward generalization of the 2-element case, we can observe that the

Figure 3. More examples of CSPs and CCSPs.

BIPARTITE INDEPENDENT SET. We say that a graph is bipartite if the vertices can be partitioned into two classes X and Y such that every edge connects a vertex of X and a vertex of Y . In the Bipartite Independent Set problem, we are looking for an independent set containing exactly k_x vertices of X and k_y vertices of Y . This problem is equivalent to a CCSP over the domain $\{0_x, 0_y, 1_x, 1_y\}$ where each edge is represented by the binary relation

$$R_{BIS} = \begin{pmatrix} 0_x & 0_x & 1_x \\ 0_y & 1_y & 0_y \end{pmatrix}$$

and we require k_x variables with value 1_x and k_y variables with value 1_y in the solution. Bipartite Independent Set is known to be NP-hard. The variant of the problem, where we require an independent set of size k in a bipartite graph (without specifying the number of vertices in each class) is polynomial-time solvable; however, this variant cannot be expressed as a CCSP.

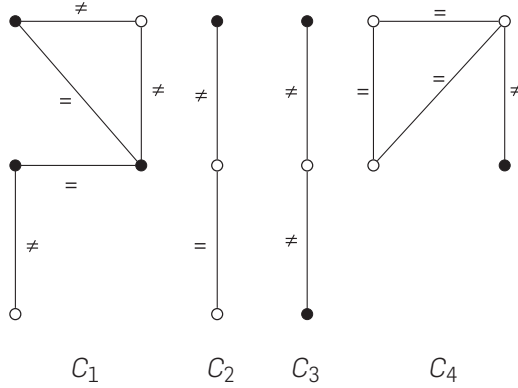
LINEAR EQUATIONS. In the regular Linear Equations problem the question is, given a system of linear equations over a finite field, decide whether it is consistent or not. The version of this problem allowing global cardinality constraints asks whether such a system has a solution that assigns each of the elements from the field to a prescribed number of variables. While Linear Equations without cardinality constraints is polynomial-time solvable, cardinality constraints make it NP-complete,⁵ even if the variables are over the two element field and every equation is of the form $x + y + z = 1$. This means that CCSP($\{R_{ODD-3}\}$) is NP-complete, where

$$R_{ODD-3} = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

is the ternary relation satisfied by an odd number of 1s.

Figure 4. Using dynamic programming to solve Boolean CCSP with binary equalities and disequalities.

Example 1. Let $\Gamma = \{=, \neq\}$ contain the binary equality and disequality relations. Consider the following instance of $\text{CCSP}(\Gamma)$ with 15 variables and 13 constraints:



Each component has exactly two satisfying assignments: either the “black” variables have value 0 and the “white” variables have value 1, or vice versa. Let set Π_i contain all possible pairs (x, y) such that the union of the first i components have a solution with x 0's and y 1's. Then

$$\begin{aligned}\Pi_1 &= \{(2, 3), (3, 2)\} \\ \Pi_2 &= \{(3, 5), (4, 4), (5, 3)\} \\ \Pi_3 &= \{(4, 7), (5, 6), (6, 5), (7, 4)\} \\ \Pi_4 &= \{(5, 10), (6, 9), (7, 8), (8, 7), (9, 6), (10, 5)\}\end{aligned}$$

If component C_i has b_i black and w_i white vertices, then clearly a pair (x, y) is in C_i if and only if either $(x - b_i, y - w_i) \in \Pi_{i-1}$ or $(x - w_i, y - b_i) \in \Pi_{i-1}$. This gives us an efficient way of computing Π_i if Π_{i-1} has been computed.

problem is polynomial-time solvable if every relation can be expressed by graphs of bijective mappings. For a mapping $\varphi: A \rightarrow A$, the *graph* of φ is the binary relation consisting of pairs of the form $(a, \varphi(a))$, $a \in A$. In this case, setting a single value in a component uniquely determines all the values in the component. Therefore, if the domain is D , then there are at most $|D|$ possible assignments in each component, and the same dynamic programming technique can be applied (but this time the set Π_i contains $|D|$ -tuples instead of pairs).

One might be tempted to guess that the class described in the previous paragraph is the only class where CCSP is polynomial-time solvable. However, it turns out that there are more general tractable classes. First, suppose that the domain is partitioned into equivalence classes, and the binary constraints are mappings between the sets of equivalence classes. This means that the values in the same equivalence class are completely interchangeable. Thus it is sufficient to keep one representative from each class, and then the problem can be solved by the algorithm sketched in the previous paragraph. Again, one might believe that this construction gives all the tractable classes, but the example in Figure 5 shows that there are more complicated constraint languages, where CCSP

is polynomial-time solvable, but we have to do two-level dynamic programming on the subcomponents of each component. It is not difficult to make this example more complicated in such a way that we have to look at sub-subcomponents and perform multiple levels of dynamic programming. This suggests that it would be difficult to characterize the tractable relations in a simple combinatorial way.

2.3. Algorithm for the tractable CCSP problems

In this section, we present a general algorithm for solving CCSP. We prove our dichotomy theorem by showing that for every finite constraint language Γ , either this algorithm solves $\text{CCSP}(\Gamma)$ in polynomial time, or $\text{CCSP}(\Gamma)$ is NP-complete. In this section, we cannot give a full characterization of those constraint languages Γ for which the algorithm works: we postpone it to Section 3.3, as it can be done most conveniently using the algebraic tools introduced in the next section.

The first condition that we require is that every relation in Γ is defined by its binary projections. Formally, we say that r -ary relation R is *2-decomposable*, if there are binary relations R_{ij} ($1 \leq i < j \leq r$) such that $(a_1, \dots, a_r) \in R$ if and only if $(a_i, a_j) \in R_{ij}$ for every $1 \leq i < j \leq r$. For example, the relation R in Figure 5 is 2-decomposable, as it is shown by the relations

$$R_{12} = \begin{pmatrix} 1 & 1 & a & d \\ 2 & 4 & b & e \end{pmatrix} \quad R_{13} = \begin{pmatrix} 1 & 1 & a & d \\ 3 & 5 & c & c \end{pmatrix} \\ R_{23} = \begin{pmatrix} 2 & 4 & b & e \\ 3 & 5 & c & c \end{pmatrix}.$$

On the other hand, relation $R_{\text{odd-3}}$ of Figure 3 is *not* 2-decomposable: all three of the corresponding relations R_{12}, R_{13}, R_{23} contain the pair $(0, 0)$, but tuple $(0, 0, 0)$ is not in R .

If a constraint is 2-decomposable, then it can be expressed by a set of binary constraints. Thus in the following, we can assume that every constraint of the CCSP instance is binary.

The algorithm finds all cardinality constraints that are satisfied by solutions of the instance. First, given an instance, we make sure that every variable v is associated with a domain D_v that contains all the values that are useful for this variable. That is, if $\langle (v, w), R \rangle$ is a constraint, then D_v is exactly $\{x \mid (x, y) \in R\}$, or in other words, D_v is exactly the set of values that the pairs of R contain at the position corresponding to v . This is achieved by the standard propagation algorithm, see, e.g., Freuder¹⁹.

A binary constraint $\langle (v, w), R \rangle$ is *trivial* if $R = D_v \times D_w$, allowing any combination of values from the domains of v and w . Let G be the graph formed by the *nontrivial* binary constraints of the problem. If graph G is disconnected, then arbitrary satisfying assignments for the connected components can be combined to obtain a satisfying assignment for the instance. Therefore, the algorithm recurses on the problems induced by connected components, and then merges the solutions using the same dynamic programming approach as for Boolean CCSP (Figure 4). If G is connected, the algorithm chooses an arbitrary variable v and tries to substitute every possible value of D_v into v . This way, we get $|D_v|$ new instances and it is clear that the original problem has a solution satisfying a cardinality

Figure 5. A two-level dynamic programming algorithm for CCSP.

Example 2. We claim that $\text{CCSP}(\{R\})$ is polynomial-time solvable for the relation

$$R = \begin{pmatrix} 1 & 1 & a & d \\ 2 & 4 & b & e \\ 3 & 5 & c & c \end{pmatrix}.$$

Consider the graph on the variables where two variables are connected if and only if they appear together in a constraint. As in Figure 4, for each component, we compute a set containing all possible cardinality vectors, and then use dynamic programming. In each component, we have to consider only two cases: either every variable is in $\{1, 2, 3, 4, 5\}$ or every variable is in $\{a, b, c, d, e\}$. If every variable of component K is in $\{1, 2, 3, 4, 5\}$, then R can be expressed by the unary constant relation 1, and the binary relation $R' = \{(2, 3), (4, 5)\}$. The binary relations partition component K into sub-components $K_1, \dots, K_r$. Since R' is the graph of a mapping, there are at most 2 possible assignments for each sub-component. Thus we can use dynamic programming to compute the set of all possible cardinality vectors on K that use only the values in $\{1, 2, 3, 4, 5\}$. If every variable of K is in $\{a, b, c, d, e\}$, then R can be expressed as the unary constant relation c and the binary relation $R'' = \{(a, b), (d, e)\}$. Again, binary relation R'' partitions K into sub-components, and we can use dynamic programming on them. Observe that the sub-components formed by R' and the sub-components formed by R'' can be different: in the first case, u and v are adjacent if they appear in the second and third coordinates of a constraint, while in the second case, u and v are adjacent if they appear in the first and second coordinates of a constraint.

constraint if and only if one of the new instances has such a solution. Thus in this case, the problem can be solved by recursively solving $|D_v|$ instances and taking the union of the set of cardinality constraints satisfied by these instances.

There is no question that the scheme described above finds every cardinality constraint satisfied by the instance. The only issue is whether the running time is polynomial: branching into $|D_v|$ directions in the case when G is connected can create an exponentially large recursion tree. We identify a useful special case that guarantees a polynomial bound on the size of the recursion tree. After substituting a value into v , we can rerun the propagation algorithm to reduce the domains of the variables by throwing away those values that are no longer useful. The key property that we require is the following:

Key Property: If G is connected, then no matter what value we substitute, propagation strictly decreases the domain of *every* variable.

If this property is true, then the algorithm has to terminate after at most $|D|$ substitutions, and therefore the height of the recursion tree is at most $|D|$, which is constant for a fixed constraint language. This gives us a polynomial bound on the size of the recursion tree.

Are there constraint languages Γ for which the key property described above holds? Yes, there are, for example, if every binary relation is the graph of a bijective mapping and G is connected, then substituting any value to a variable v decreases the domain of every other variable to a single element. As mentioned earlier, it is not easy to give a simple

combinatorial characterization of those sets Γ for which the algorithm works (in the next section, we characterize them in a more algebraic way). We can at least give some necessary conditions that show what kind of generalizations of mappings should we deal with.

Let R be a binary relation from a set A to set B , that is, $R \subseteq A \times B$. Relation R is said to be a *thick mapping* if whenever pairs $(a, c), (a, d), (b, c)$ belong to R , the pair (b, d) also belongs to R . As is easily seen, any thick mapping R has two associated equivalence relations α and β on A and B , respectively, such that R can be thought of as a mapping from the set of equivalence classes of α to that of β .

To give some intuition why it is a problem if a relation is not a thick mapping, consider the relation $R = \{(a, c), (a, d), (b, c)\}$. Suppose that there are only two variables v, w and there is a single constraint $\langle (v, w), R \rangle$. In this case, the domains are $D_v = \{a, b\}$ and $D_w = \{c, d\}$. The constraint is nontrivial, thus the graph G is connected. But if we assign value a to variable v , then the domain size of w does not decrease: b and d are both possible. Thus for this relation, the algorithm does not have the property that every substitution decreases every domain, and we cannot guarantee a polynomial bound on the recursion tree.

Unfortunately, requiring that every relation is a thick mapping is not sufficient for tractability, as thick mappings can interact with each other in a way that makes CCSP hard. Therefore in order to the problem $\text{CCSP}(\Gamma)$ for a set Γ of thick mappings to be easy, more restrictions have to be imposed on Γ . Such a condition called *noncrossing* requires that if two thick mappings induce equivalence relations α and β on a certain set, then for any equivalence class C of α and a class D of β that are not disjoint, either $C \subseteq D$ or $D \subseteq C$. We need even stronger conditions: not only relations from Γ must be noncrossing thick mappings, but also certain relations derived from them. A detailed explanation is given in the next section.

3. ALGEBRAIC APPROACH

One of the main difficulties in studying problems $\text{CSP}(\Gamma)$ and $\text{CCSP}(\Gamma)$ is: How can one describe or characterize a constraint language (possibly infinite)? A combinatorial characterization is very often impossible, so two alternative approaches have been widely used, one through logic and another one through algebra. Here we use the algebraic one.

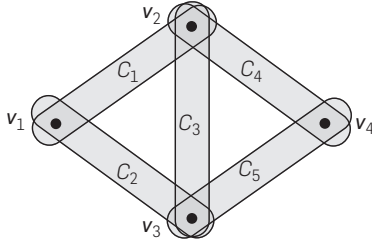
3.1. Primitive positive definitions

In a CSP, possible combinations of values of certain variables can be constrained even if there is no explicit constraint imposed on them, see Figure 6. That is, we can use the constraints in Γ to build “gadgets” that enforce a constraint relation on a certain set of variables. Note that, as in Figure 6, the constraint relation expressed by the gadget does not necessarily belong to Γ . This means that for every constraint language Γ , there is a set of *implicit* constraints that do not belong to Γ , but can still be expressed by instances of $\text{CSP}(\Gamma)$.

How can we characterize all the implicit constraints of a constraint language Γ ? It turns out that the implicit constraints that can be expressed in instances of $\text{CSP}(\Gamma)$ admit a simple logic representation. Treating relations in Γ as predicates, one can construct logic formulas from them, and use

Figure 6. Implicit constraints.

*Example 3.*⁸ Let Γ be a constraint language containing a single binary relation R over the set $D = \{0, 1, 2\}$, where R is given by $R = \{(0, 0), (0, 1), (1, 0), (1, 2), (2, 1), (2, 2)\}$. Consider the instance of $\text{CSP}(\Gamma)$ with the set of variables $\{v_1, v_2, v_3, v_4\}$ and set of constraints $\{C_1, C_2, C_3, C_4, C_5\}$, where $C_1 = \langle (v_1, v_2), R \rangle$, $C_2 = \langle (v_1, v_3), R \rangle$, $C_3 = \langle (v_2, v_3), R \rangle$, $C_4 = \langle (v_2, v_4), R \rangle$, $C_5 = \langle (v_3, v_4), R \rangle$. There is no explicit constraint on the pair (v_1, v_4) . However, by considering all solutions to the instance, it can be shown that the possible pairs of values which can be taken by this pair of variables are precisely the elements of the relation $R' = R \cup \{(1, 1)\}$. Thus this instance can be considered as a “gadget” implementing R' using only the relations R .



The relation R' can be expressed as the following primitive positive (pp-) definition:

$$R'(x, y) = \exists z, t (R(x, z) \wedge R(x, t) \wedge R(z, t) \wedge R(z, y) \wedge R(t, y)).$$

these formulas to express other predicates (relations). The type of formulas that is just right for representing implicit constraints is called *primitive positive*. Primitive positive (pp-) formulas include predicates from Γ (atomic formulas) and the equality, conjunctions of atomic formulas, and existential quantifiers. Relations (or predicates) that can be expressed by using pp-formulas with predicates from Γ are said to be *pp-definable* in Γ .

Jeavons et al.²³ proved that pp-definitions give rise to reductions between CSPs: If Γ and Δ are constraint languages on the same set such that Δ is finite and every relation in Δ is pp-definable in Γ , then $\text{CSP}(\Delta)$ is polynomial-time reducible to $\text{CSP}(\Gamma)$ (can be improved to logarithmic-space reducibility). Thus, when proving hardness of CSPs one can use any relations pp-definable in the given constraint language. Very often ‘gadgets’ used in complexity proofs can be expressed as pp-definitions, so primitive positive definitions generalize and unify gadget reductions.

In CSPs with cardinality constraints, it is not obvious that adding pp-definable relations to the constraint language does not increase hardness. The difficulty is that introducing gadgets (like the one in Figure 6) means adding auxiliary variables, and the values appearing on these variables can affect the cardinality constraints. Nevertheless, we can show that adding a new constraint R' to the constraint language of a CSP with cardinality constraints does not change the complexity if R' is pp-definable without using the equality relation. Relations expressible in such a weaker way are called *pp-definable without equality*. In fact, relations that are pp-definable in a certain Γ with or without equality can only be different by certain redundant parts that are not so important for constraint problems. Therefore, we can essentially assume

that Γ is closed under pp-definitions, and hence we can use the algebraic framework discussed in more detail in the next section.

3.2. Polymorphisms and invariants

Although pp-definitions are helpful in hardness proofs, they do not resolve the main difficulty of studying the complexity of CSPs, as they do not help much in describing constraint languages. However, pp-definitions provide a bridge to a tool that allows to do that. *Polymorphisms* can be viewed as a sort of extended symmetries of relations. Let R be a relation on some set D and f a function on the same set that may depend on more than one variable; let f be n -ary, that is, depends on n variables. The function f is a polymorphism of R if for any choice of tuples $\bar{a}_1, \dots, \bar{a}_n$ from R the tuple $f(\bar{a}_1, \dots, \bar{a}_n)$ obtained by component-wise application of f also belongs to R . Relation R in this case is said to be an *invariant* of f . Polymorphisms and invariants naturally extend to constraint languages and functions: A function is a polymorphism of a constraint language if it is a polymorphism of every relation in it, and a relation is an invariant of a set of functions if it is an invariant of every function in the set. For constraint languages Γ , and set of functions C , by $\text{Pol } \Gamma$ we denote the set of all polymorphisms of Γ , and $\text{Inv } C$ the set of all invariants of C , see Figure 7.

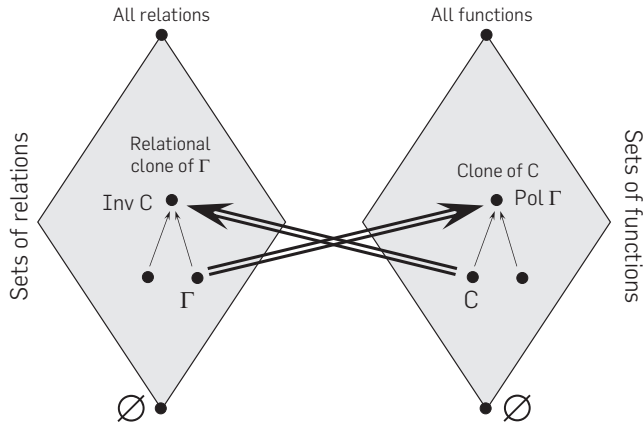
Sets of the form $\text{Pol } \Gamma$ and $\text{Inv } C$ have a number of interesting properties, see, e.g., Denecke and Wismath.¹⁶ For any set C of functions $\text{Inv } C$ is a *relational clone*, that is, constraint language Δ such that every relation pp-definable in Δ also belongs to Δ . Therefore Jeavons’ result (and this paper’s analogous result) can be stated in terms of polymorphisms: If Γ and Δ are constraint languages on the same set such that Δ is finite and every polymorphism of Γ is also a polymorphism of Δ , then $\text{CSP}(\Delta)$ is polynomial-time reducible to $\text{CSP}(\Gamma)$. For CCSP we only have to add the requirement that relations in Δ do not contain redundancies.

For any constraint language Γ the set $\text{Pol } \Gamma$ is a *clone*, that is, a set of functions that contains the identity functions, and closed under compositions. Clones have been a subject of intensive study in algebra for decades; the results of those studies are readily available to be applied to constraint problems.

Clearly, large constraint languages have few polymorphisms. Thus, a number of important properties of relations can be inferred merely from the existence of polymorphisms of certain types. A ternary function h on a set D is said to be a *majority function* if $h(x, x, y) = h(x, y, x) = h(y, x, x) = x$ for any $x, y \in D$. If a constraint language has a polymorphism that is a majority function, then the constraint language is 2-decomposable. A ternary operation m is called Maltsev if $m(x, y, y) = m(y, y, x) = x$ for any $x, y \in D$. Any binary relation having a Maltsev polymorphism is a thick mapping, see Figure 8.

For regular CSPs, complexity questions are usually reduced one step further, to universal algebras and their varieties. Most of the strong complexity results about CSPs are obtained this way.^{3, 7, 9} Moreover, research on CSP complexity have revolutionized certain fields of algebra, see, e.g., Barto and Kozik.⁴ For our result, however, we do not need more algebra than polymorphisms.

Figure 7. Pol and Inv.



3.3. Easy cardinality constraints: The full result

We can finally explain the main result in details. A function f is said to be *conservative* if it always equal to one of its arguments. For instance, a ternary function f is conservative if $f(a, b, c) \in \{a, b, c\}$ for any a, b, c . The main result can be stated compactly the following way:

Main Theorem *Let Γ be a finite constraint language. If Γ has a majority polymorphism and has a conservative Maltsev polymorphism, then $\text{CCSP}(\Gamma)$ is polynomial-time solvable. Otherwise, the problem is NP-complete.*

We can show that if a constraint language Γ satisfies the conditions above, then the problem can be solved in polynomial time by the algorithm presented in Section 2.3. Let Δ be the set of binary relations pp-definable in Γ . Since Γ has a majority polymorphism, it is 2-decomposable; hence, every constraint with a relation $R \in \Gamma$ can be replaced with a collection of binary constraints, the ‘projections’ of R , which are pp-definable in Γ and thus belong to Δ . Therefore we only need to verify that the Key Property (Section 2.3) always holds. Due to 2-decomposability, Γ can be replaced with Δ . This constraint language has a Maltsev polymorphism, and this makes its relations thick mappings. Suppose now that the graph G of a problem from $\text{CCSP}(\Delta)$ is connected. For any two variables v, w the set of all allowed combinations of their values is a binary relation, denoted R_{vw} and an implicit constraint. Since Δ contains all binary relations pp-definable in Δ , we have $R_{vw} \in \Delta$. Thus R_{vw} is a thick mapping from D_v to D_w . The connectedness of G and the fact that all relations in Δ are noncrossing can be used to show that R_{vw} is a nontrivial thick mapping. Let α and β be equivalence relations it induces on D_v and D_w , respectively. If we fix a value $a \in D_v$ then the possible values of w are restricted to one equivalence class of β , a proper subset of D_w . As this is true for all variables w , the Key Property follows.

The Main Theorem also leads to a more combinatorial characterization of tractable problems $\text{CCSP}(\Gamma)$: Such a problem is tractable if and only if Γ is 2-decomposable, and the binary relations pp-definable in Γ are noncrossing thick mappings.

Figure 8. Examples of polymorphism.

Majority implies 2-decomposability.

Let R be a ternary relation and h a majority function, which is a polymorphism of R . We show that any triple (a, b, c) such that each of (a, b) , (b, c) , and (a, c) is extendable to a triple from R , belongs to R . This means the 2-decomposability of R in this case. By the assumption, there are (a, b, z) , (a, y, c) , $(x, b, c) \in R$ for some x, y, z . Since h is a majority polymorphism of R we have

$$h \begin{pmatrix} a & a & x \\ b & y & b \\ z & c & c \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix},$$

and (a, b, c) belongs to R .

Maltsev implies thick mapping.

Let R be a binary relation and m its Maltsev polymorphism. We have to prove that for any (a, c) , (a, d) , $(b, c) \in R$ the pair (b, d) also belong to R . It follows from a single application of the Maltsev polymorphism:

$$m \begin{pmatrix} a & a & b \\ d & c & c \end{pmatrix} = \begin{pmatrix} b \\ d \end{pmatrix}.$$

Linear equations.

As another example of a property of relations expressible by a polymorphism, we consider relations that are solution spaces of systems of linear equations over a finite field F . Then if a relation R has such representation it is an invariant of the affine function $f(x, y, z) = x - y + z$, where $+$, $-$ are operations of the field F . Indeed, let $A \cdot \mathbf{x} = \mathbf{b}$ be the system defining R , and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in R$. Then

$$A \cdot f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = A \cdot (\mathbf{x} - \mathbf{y} + \mathbf{z}) = A \cdot \mathbf{x} - A \cdot \mathbf{y} + A \cdot \mathbf{z} = \mathbf{b}.$$

In fact, the converse can also be shown: if R is invariant under f then it is the solution space of a certain system of linear equations.

What remains now is to show that otherwise the problem is hard.

4. HARD CSPS WITH CARDINALITY CONSTRAINTS

If one of the three conditions on a constraint language Γ (a) 2-decomposability, (b) all binary pp-definable relations are thick mappings, and (c) all such binary relations are noncrossing does not hold, we show that either Bipartite Independent Set or Linear Equation is reducible to $\text{CCSP}(\Gamma)$, thus showing that $\text{CCSP}(\Gamma)$ is NP-complete. This part is technical, but we outline the intuition behind the technique.

Suppose first that a binary relation R is pp-definable in Γ , but is not a thick mapping. This means that for some a, b, c, d pairs (a, c) , (a, d) , (b, c) belong to R while (b, d) does not. If a, b, c, d are distinct values, then R contains a fragment that looks like R_{BIS} . We exploit this fact to reduce Bipartite Independent Set to $\text{CCSP}(\Gamma)$ and conclude NP-hardness in this case. In general, it is possible that some of a, b, c, d coincide. However, a case analysis shows that reduction from Bipartite Independent Set is possible in all cases.

If there exist two thick mappings pp-definable in Γ that are not noncrossing, then there are also two equivalence relations with this property; denote them α and β . Since they

are not noncrossing, some α -class and some β -class overlap, but are not subsets of one another. Hence for some a, b, c , we have (a, b) is in α but not in β , and (b, c) is in β but not in α . If we can restrict α and β onto $\{a, b, c\}$ somehow, then the product of binary relations $\alpha \circ \beta$ given by a pp-formula $\exists z \alpha(x, z) \wedge \beta(z, y)$, contains (a, a) , (a, c) , (c, c) , but does not contain (c, a) . Again, this fact can be used to reduce Bipartite Independent Set to CCSP(Γ).

Finally, let $R \in \Gamma$ be non-2-decomposable. For simplicity assume R ternary. There is a triple (a, b, c) such that (a, b, z) , (a, y, c) , (x, b, c) belong to R for some x, y, z , but (a, b, c) does not. We show that either a binary relation which is not a thick mapping can be pp-defined in Γ , or two thick mappings that are not noncrossing, or all the tuples can be chosen such that $a = b = c = 0$, $x = y = z = 1$ (we assume 0 and 1 are elements of the domain we can use here), and R restricted to $\{0, 1\}$ is $R_{\text{odd-3}}$. Therefore a reduction of Linear Equations to CCSP(Γ) can be found.

5. CONCLUSION

We have completed the study of CSP extended with cardinality constraints, and proved a dichotomy theorem characterizing the complexity of the problem for every constraint language Γ over an arbitrary finite domain D . Dichotomy theorems over non-Boolean domains are notoriously hard to prove, but possibly due to the rather restrictive nature of the CCSP problem, we managed to obtain a complete characterization. One can think of several natural variants with more expressive power, for example, the domain is $\{1, 2, 3, 4\}$, and we have upper bounds on the cardinalities of 1 and 2, while there are lower bounds on the cardinalities of 3 and 4. Therefore, upper and/or lower bounds instead of exact cardinality requirements, bounds only on a subset of values, bounds on the total cardinality of a subset of values, etc. give lots of interesting problems to look at. However, some of these questions seem to be very difficult, as a dichotomy result would immediately imply the Feder–Vardi Dichotomy Conjecture (after all, we do not fully understand CSP even without cardinality constraints).

Another natural direction is to consider optimization variants (minimize/maximize the number of times certain values appear) and determine the approximability of the resulting problems. In the Boolean case, the approximability of the MinOnes/MaxOnes problems, where the task is to find a satisfying assignment minimizing/maximizing the number of variables receiving value 1, was classified by Khanna et al.²⁵ Again, not being able to solve the Feder–Vardi conjecture limits what immediate progress we can expect in the study of non-Boolean domains.

Finally, one can look at CCSP from the viewpoint of parameterized complexity. The basic issues of parameterized complexity is whether an algorithm of running time $f(k) \cdot n^c$ exists, where k is some parameter of the input (for example, the size of the solution we are looking for), $f(k)$ is an arbitrary function depending on k , and c is a universal constant independent of k . For example, in Boolean CCSP, one can answer in time $n^{O(k)}$ whether there is a solution with exactly k variables set to 1, but it would be preferable to find an algorithm with running time of the form $f(k) \cdot n^c$, that is,

where the combinatorial explosion is restricted to k and the exponent of n is independent of k . We can ask what those Boolean constraint languages Γ are for which the problem of finding a solution with exactly/at most/at least k variables having 1 can be solved in such running time. These questions have been investigated and completely answered in Kratsch et al. and Marx.^{27, 30} Generalization of some of these results to arbitrary non-Boolean domains have been obtained very recently by the authors.¹¹ □

References

- Allen, J. *Natural Language Understanding*. Benjamin Cummings, 1994.
- Allender, E., Bauland, M., Immerman, N., Schnoor, H., Vollmer, H. The complexity of satisfiability problems: Refining Schaefer's theorem. *J. Comput. Syst. Sci.* 75, 4 (2009), 245–254.
- Barto, L., Kozik, M. Constraint satisfaction problems of bounded width. In *FOCS* (2009), 595–603.
- Barto, L., Kozik, M. New conditions for Taylor varieties and CSP. In *LICS*, 2010. to appear.
- Bazgan, C., Karpinski, M. On the complexity of global constraint satisfaction. In *ISAAC* (2005), 624–633.
- Bessière, C., Hebrard, E., Hnich, B., Walsh, T. The complexity of global constraints. In *AAAI* (2004), 112–117.
- Bulatov, A. Tractable conservative constraint satisfaction problems. In *LICS* (2003), 321–330.
- Bulatov, A., Jeavons, P., Krokhin, A. Functions of multiple-valued logic and the complexity of constraint satisfaction: A short survey. In *ISMVL* (2003), 343–351.
- Bulatov, A.A. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *J. ACM* 53, 1 (2006), 66–120.
- Bulatov, A.A., Krokhin, A.A., Larose, B. Dualities for constraint satisfaction problems. In *Complexity of Constraints* (2008), 93–124.
- Bulatov, A.A., Marx, D. Constraint satisfaction parameterized by solution size. Manuscript.
- Bulatov, A.A., Marx, D. The complexity of global cardinality constraints. In *LICS* (2009), 419–428.
- Bulatov, A.A., Valeriote, M. Recent results on the algebraic approach to the CSP. In *Complexity of Constraints* (2008), 68–92.
- Creignou, N., Schnoor, H., Schnoor, I. Non-uniform boolean constraint satisfaction problems with cardinality constraint. In *CSL* (2008), 109–123.
- Dechter, R. *Constraint Processing*. Morgan Kaufmann Publishers, 2003.
- Denecke, K., Wismath, S. *Universal Algebra and Applications in Theoretical Computer Science*. Chapman and Hall, CRC Press, 2002.
- Dunkin, N., Bater, J., Jeavons, P., Cohen, D. Toward high order constraint representations for the frequency assignment problem. Technical Report CSD-TR-98-05, Department of Computer Science, Royal Holloway, University of London, Egham, Surrey, UK, 1998.
- Feder, T., Vardi, M. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.* 28 (1998), 57–104.
- Freuder, E. Synthesizing constraint expressions. *Commun. ACM* 21, (1978) 958–966.
- Freuder, E.C. Complexity of k-tree structured constraint satisfaction problems. In *Proceedings of AAAI-90* (Boston, MA, 1990), 4–9.
- Grohe, M. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM* 54, 1 (2007).
- Hell, P., Nesetril, J. Colouring, constraint satisfaction, and complexity. *Comput. Sci. Rev.* 2, 3 (2008), 143–163.
- Jeavons, P., Cohen, D., Gyssens, M. Closure properties of constraints. *J. ACM* 44 (1997), 527–548.
- Kautz, H.A., Selman, B. Planning as satisfiability. In *ECAI* (1992), 359–363.
- Khanna, S., Sudan, M., Trevisan, L., Williamson, D.P. The approximability of constraint satisfaction problems. *SIAM J. Comput.* 30, 6 (2001) 1863–1920.
- Kolaitis, P., Vardi, M. Conjunctive query containment and constraint satisfaction. *J. Comput. Syst. Sci.* 61 (2000), 302–332.
- Kratsch, S., Marx, D., Wahlström, M. Parameterized complexity and kernelizability of Max Ones and Exact Ones problems. Submitted, 2010.
- Ladner, R. On the structure of polynomial time reducibility. *J. ACM* 22 (1975), 155–171.
- Larose, B., Tesson, P. Universal algebra and hardness results for constraint satisfaction problems. In *ICALP* (2007), 267–278.
- Marx, D. Parameterized complexity of constraint satisfaction problems. *Comput. Complex.* 14, 2 (2005), 153–183. Special issue “Conference on Computational Complexity (CCC) 2004”.
- Marx, D. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *STOC* (2010), to appear.
- Montanari, U. Networks of constraints: Fundamental properties and applications to picture processing. *Inf. Sci.* 7 (1974), 95–132.
- Nadel, B. Constraint satisfaction in Prolog: Complexity and theory-based heuristics. *Inf. Sci.* 83, 3–4 (1995), 113–131.
- Schaefer, T. The complexity of satisfiability problems. In *STOC* (1978), 216–226.
- van Beek, P. Reasoning about qualitative temporal information. *Artif. Intell.* 58 (1992), 297–326.

Andrei A. Bulatov (abulatov@cs.sfu.ca), School of Computing Science, Simon Fraser University, 8888 University Drive, Burnaby, BC, Canada.

Dániel Marx (dmarx@cs.bme.hu), School of Computer Science, Tel Aviv University, Tel Aviv, Israel.

Air Force Institute of Technology (AFIT) Dayton, Ohio

Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Faculty Positions in Computer Science or Computer Engineering

The Department of Electrical and Computer Engineering is seeking applicants for tenure track positions in computer science or computer engineering. The department is particularly interested in receiving applications from individuals with strong backgrounds in formal methods (with emphasis on cryptography), software engineering, bioinformatics, computer architecture/VLSI systems, and computer networks and security. The positions are at the assistant professor level, although qualified candidates will be considered at all levels. Applicants must have an earned doctorate in computer science or computer engineering or closely related field and must be U.S. citizens. These positions require teaching at the graduate level as well as establishing and sustaining a strong research program.

AFIT is the premier institution for defense-related graduate education in science, engineering, advanced technology, and management for the U.S. Air Force and the Department of Defense (DoD). Full details on these positions, the department, and application procedures can be found at: http://www.afit.edu/en/eng/employment_faculty.cfm

Review of applications will begin immediately and will continue until the positions are filled. The United States Air Force is an equal opportunity, affirmative action employer.

Cal Poly State University **Tenure Track Position - Forbes Professor of Computer Engineering**

COMPUTER ENGINEERING - The Computer Science Department and Computer Engineering Program at Cal Poly, San Luis Obispo, invite applications for a full-time, academic year tenure-track Computer Engineering faculty position at the Assistant or Associate Professor rank, beginning no later than Fall 2011. The appointment will be designated as the "Forbes Professor of Computer Engineering". Duties include teaching core undergraduate courses, and upper-division and master's level courses in a specialty area; performing research in an area of computer engineering; and service to the department, the university, and the community.

Applicants from all mainstream areas of computer engineering are encouraged to apply. A doctorate in Computer Engineering, Computer Science, Electrical Engineering, or a closely related field is required. Salary is commensurate with qualifications and experience.

Candidates in the areas of: Computer Security,

Parallel and Distributed Computing, Autonomous Systems, Biomedical Applications, and Sustainable Computing are strongly encouraged to apply. Industrial experience and willingness to teach in multiple areas of the undergraduate curriculum are desirable.

Candidates must have a strong commitment to teaching excellence and laboratory-based instruction; dedication to continued professional development and scholarship; and a broad-based knowledge of computer engineering. Demonstrated ability in written and oral use of the English language is required.

Cal Poly offers Bachelor's Degrees in Computer Engineering, Computer Science, Software Engineering and Electrical Engineering, and Master's Degrees in Computer Science and Electrical Engineering. Computer Engineering is a joint program between the Departments of Computer Science and Electrical Engineering. Cal Poly emphasizes "learn by doing" which involves extensive lab work and projects in support of theoretical knowledge. The available computing facilities for instructional and faculty support are modern and extensive.

Apply: <http://www.calpolyjobs.org>, apply to requisition 102122

Central Michigan University **Department of Computer Science**

The department has two tenure track positions in Information Technology to be filled in Fall 2011. One is in Applied Networking and one is in Medical or Health Informatics. For more information and to apply electronically: www.jobs.cmich.edu.

Furman University **Assistant Professor of Computer Science**

The Department of Computer Science invites applications for a tenure track position at the Assistant Professor level to begin in the fall of 2011. Candidates must have a Ph.D. in Computer Science or a closely related field. The position requires teaching excellence, effective institutional service, and an ability to work with colleagues across disciplines. An ability to develop a program of scholarly and professional activity involving undergraduates is a priority. Research specialty areas being sought include (but are not limited to) high performance computing, computational science, mathematical modeling, and bioinformatics. Of particular interest are candidates willing to engage in collaborative research that bridges the computational and medical sciences. The position will be initially funded by and is expected to contribute to a major multi-disciplinary and multi-organizational state-wide initiative aimed at biofabrication of tissues and organs.

Furman is a highly selective, independent, top 40 undergraduate liberal arts institution with an enrollment of approximately 2600 students. The

university is located in the vibrant and beautiful upstate region of South Carolina, offers generous benefits to fulltime faculty, and subscribes to a problem-solving, project-oriented, experience-based approach to education that is referred to as Engaged Learning. The Department of Computer Science confers the B.S. degree with majors in Computer Science, Information Technology, and Computer Science/Mathematics. The successful candidate will have the opportunity to teach in Furman's First Year Seminar program. Furman University is an equal-opportunity employer. Women and underrepresented minorities are strongly encouraged to apply. For the complete ad, please visit <http://cs.furman.edu>.

Applicants should submit a curriculum vitae, statement of teaching philosophy, description of research interests, an official copy of most recent transcripts, and have three letters of recommendation sent separately. Please send all materials to Dr. Kevin Treu, Chair, Department of Computer Science, Furman University, 3300 Poinsett Hwy, Greenville, SC 29613. Materials may also be sent in PDF format to kevin.treu@furman.edu. Review of applications will continue until the position is filled.

Harvard University **Tenure-track Faculty Position in Biorobotics**

The School of Engineering and Applied Sciences (SEAS) and the Wyss Institute for Biologically Inspired Engineering at Harvard University (Wyss Institute) seek applicants for a tenure-track faculty position. The position will be at the level of assistant professor in SEAS in the field of Biorobotics. Potential subareas include, but are not limited to:

- Medical robots (e.g. prosthetics and rehabilitation robotics)
- Robot locomotion (e.g. animal-inspired robotic systems, bio-inspired adaptive locomotion and control)
- Dynamics and control (e.g. machine learning and robotics, swarm and modular robotics, and human-robot interaction)
- Sensors and actuators (e.g. novel electroactive materials)
- MEMS/NEMS devices and robots
- New concepts for energy storage
- Biomimetic materials for robotics

In addition to having a faculty appointment in SEAS, the successful candidate will also become a core faculty member of the Wyss Institute, which is composed of engineers, scientists, clinicians and theoreticians from Harvard, its affiliated hospitals, and other leading academic institutions in the Boston/Cambridge region. The Wyss Institute focuses on fundamental science-driven technology development in the field of Biologically Inspired Engineering.

For additional information, visit the following Websites:

SEAS: <http://www.seas.harvard.edu/>
Wyss Institute: <http://wyss.harvard.edu/>

Candidates must have the ability to develop a leading research program with a focus on technology development and translation. An enthusiasm for teaching is essential, and responsibilities will include both core undergraduate engineering courses as well as graduate-level courses.

An application, assembled as a single PDF file, should include a *curriculum vitae*, separate two-page statements of research and teaching interests, up to three scientific papers, and names and contact information for at least three writers of letters of recommendation. Applications should be sent to biorobotics_search@seas.harvard.edu. The deadline for applications is October 31, 2010.

Applications will be reviewed beginning February 2010 and will be accepted until the position is filled.

Harvard University is an Equal Opportunity/Affirmative Action Employer.

Applications from women and minority candidates are strongly encouraged.

Princeton University Computer Science, Assistant Professor Tenure-Track Positions

The Department of Computer Science at Princeton University invites applications for faculty positions at the Assistant Professor level. We are accepting applications in all areas of Computer Science.

Applicants must demonstrate superior research and scholarship potential as well as teaching ability. A PhD in Computer Science or a related area is required.

Successful candidates are expected to pursue an active research program and to contribute significantly to the teaching programs of the department. Applicants should include a resume contact information for at least three people who can comment on the applicant's professional qualifications.

There is no deadline, but review of applications will start in December 2010; the review of applicants in the field of theoretical computer science will begin as early as October 2010.

Princeton University is an equal opportunity employer and complies with applicable EEO and affirmative action regulations. You may apply online at: <http://www.cs.princeton.edu/jobs>

Requisition Number: 1000520

Santa Clara University Tenure Track position in Management Information Systems Department of Operations and Management Information Systems

The Leavey School of Business invites applications for a tenure track position in Management Information Systems within the Department of Operations and Management Information Systems beginning Fall, 2011. **Santa Clara University** is a private Jesuit university located in the heart of Silicon Valley.

We are looking for two candidates with research interest in areas of business intelligence, data mining technologies, social networking, ERP, and CRM. The successful applicant will have a Ph.D. in Management Information Systems or related field and possess superior communication skills and a commitment to excellence in both teaching and research. Santa Clara University is an equal opportunity/affirmative action employer and welcomes applications from women, persons of color, and members of historically under-represented U.S. ethnic groups. Please email (preferred) a letter of application, vita,

references and teaching evaluations to Eileen Turner at eturner@scu.edu, or

mail your application packet to the following address:

Chair
OMIS Search Committee
Santa Clara University
500 El Camino Real
Santa Clara, CA 95053-0382

University of Technology, Sydney Lecturer/Senior Lecturer

The Faculty of Engineering and Information Technology is seeking applicants for a full-time permanent lecturer/senior lecturer position to build on the existing research and teaching in the faculty in the field of game programming & development, animation techniques or closely related field. For detailed information, visit <http://www.hru.uts.edu.au/jobs/>



THE CHINESE UNIVERSITY OF HONG KONG

Applications are invited for:-

Department of Computer Science and Engineering and Department of Information Engineering Professors / Associate Professors / Assistant Professors (Ref. 1011/008(255)/2)

The Department of Computer Science and Engineering (CSE) and the Department of Information Engineering (IE) invite applications for several posts in all areas of 'cyber-security' at Assistant Professor, Associate Professor or Professor levels, including:

- system security
- network security
- theoretical and/or applied cryptography

The CSE Department and the IE Department have around 30 and 25 faculty members respectively from leading universities and research institutions. Further information about the departments is available at <http://www.cse.cuhk.edu.hk> and <http://www.ie.cuhk.edu.hk> respectively.

Applicants should have (i) a relevant PhD degree; (ii) strong commitment to excellence in research and teaching; and (iii) outstanding accomplishments and research potential. The posts are available from the academic year 2011-2012, and appointments will normally be made on contract basis for up to three years initially commencing August 2011, which, subject to performance and mutual agreement, may lead to longer-term appointment or substantiation later. Applications will be accepted until the posts are filled.

Salary and Fringe Benefits

Salary will be highly competitive, commensurate with qualifications and experience. The University offers a comprehensive fringe benefit package, including medical care, plus a contract-end gratuity for appointments of two years or longer, and housing benefits for eligible appointees. Further information about the University and the general terms of service for appointments is available at <http://www.cuhk.edu.hk/personnel>. The terms mentioned herein are for reference only and are subject to revision by the University.

Application Procedure

Please send full resume with a cover letter, a teaching statement, a research statement, together with names, addresses and fax numbers/e-mail addresses of at least three referees to whom the applicants' consent has been given for their providing references in .pdf format via e-mail to recruit@erg.cuhk.edu.hk, or recruit@cse.cuhk.edu.hk, or recruit@ie.cuhk.edu.hk. The Personal Information Collection Statement will be provided upon request. Please quote the reference number and mark 'Application - Confidential' on cover.



ARIZONA STATE UNIVERSITY Engineering Faculty opening in Human Activity Capture and Analysis

The School of Arts, Media and Engineering (AME) and the School of Electrical, Computer and Energy Engineering (ECEE) at Arizona State University are seeking a jointly appointed faculty member. Of particular interest is the area of *Human Activity Capture and Analysis* with emphasis on health, education or cultural applications. Candidates are sought at the assistant, associate or full professor level.

The School of Arts, Media and Engineering (AME - <http://ame.asu.edu>), at the Herberger Institute for Design and the Arts and the Ira Fulton Schools of Engineering, is a leading transdisciplinary program in media arts and sciences. It offers PhD, Masters and undergraduate degrees in new media in collaboration with 12 partner units spanning arts, design, sciences and engineering. Significant federal, private foundation and industry support along with clinical, education and cultural partnerships contribute to the development and deployment of innovative media systems. The School of Electrical, Computer and Energy Engineering leads academic programs with more than 50 faculty members, 500 undergraduates and 700 graduate students. The school's programs include extramural research funding of more \$20M and BSE, MSE, MS and Ph.D. degree programs. Both Schools are strongly committed to interdisciplinary research and education. **Application deadline: November 1, 2010.** For complete position details, please visit: <http://ame.asu.edu/about/employment.php>

Take Advantage of ACM's Lifetime Membership Plan!

- ◆ **ACM Professional Members** can enjoy the convenience of making a single payment for their entire tenure as an ACM Member, and also be protected from future price increases by taking advantage of **ACM's Lifetime Membership** option.
- ◆ **ACM Lifetime Membership** dues may be tax deductible under certain circumstances, so becoming a Lifetime Member can have additional advantages if you act before the end of 2009. (Please consult with your tax advisor.)
- ◆ Lifetime Members receive a certificate of recognition suitable for framing, and enjoy all of the benefits of **ACM Professional Membership**.

Learn more and apply at:
<http://www.acm.org/life>



Association for
Computing Machinery

Advancing Computing as a Science & Profession



ADVERTISING IN CAREER OPPORTUNITIES

How to Submit a Classified Line Ad: Send an e-mail to acmm mediasales@acm.org. Please include text, and indicate the issue/or issues where the ad will appear, and a contact name and number.

Estimates: An insertion order will then be e-mailed back to you. The ad will be typeset according to CACM guidelines. **NO PROOFS** can be sent. Classified line ads are **NOT** commissionable.

Rates: \$325.00 for six lines of text, 40 characters per line. \$32.50 for each additional line after the first six. The **MINIMUM** is six lines.

Deadlines: Five weeks prior to the publication date of the issue (which is the first of every month). Latest deadlines:
<http://www.acm.org/publications>

Career Opportunities Online: Classified and recruitment display ads receive a free duplicate listing on our website at:
<http://campus.acm.org/careercenter>

Ads are listed for a period of 30 days.

For More Information Contact:

ACM Media Sales
at 212-626-0686 or
acmm mediasales@acm.org



Department Head Department of Electrical Engineering & Computer Science South Dakota State University Brookings, SD

South Dakota State University invites applications and nominations for the position of Department Head of Electrical Engineering & Computer Science. SDSU, the state's land-grant and largest university, is a Carnegie RU/H (high research activity) institution with 12,400 students. The university is seeking an energetic academic leader with strategic vision, outstanding academic credentials and successful administrative experience. The Department Head, who reports to the Dean of Engineering, holds a 12-month position and oversees all of the department's administrative functions including academic, budget, facilities, research and outreach. In FY 2010 the department had 25 base-funded faculty and 390 students enrolled in undergraduate and graduate programs in electrical engineering, computer science and software engineering. The department is enjoying strong growth in enrollments and funded research, strong ties to industry and a beautiful new \$12 million-72,000 sq. ft. building.

The successful applicant must have an earned Ph.D. and distinguished record of performance consistent with appointment as a tenured full professor in a discipline appropriate to the department. He/she must also have a record of innovative and strategic leadership that would apply to a progressive and growing academic environment and a record of effective university administrative experience.

For detailed electronic application instructions, a full description of the position and information on the department, university and community, please visit <http://www.sdstate.edu/eecs/>. For the most complete consideration, applications should be received by Nov. 1, 2010. For questions on the electronic employment process, contact SDSU Human Resources at (605) 688-4128.

South Dakota State University is an AA/EEO employer.



DOI:10.1145/1810891.1810917

Peter Winkler

Puzzled Solutions and Sources

It's amazing how little we know about good old plane geometry. Last month (August 2010, p. 128) we posted a trio of brainteasers, including one as yet unsolved, concerning figures on a plane. Here, we offer solutions to two of them.

1. Covering a Gravy Stain.

Solution. The object was to cover a gravy stain of area less than one square inch with a plastic sheet containing a grid of side one inch in such a way that no intersection point of the grid fell on the stain. This puzzle (as I was reminded by Andrei Furtuna, a Dartmouth computer science graduate student) may date to the great Lithuanian-born mathematician Hermann Minkowski (1864–1909).

It suffices to consider only grids oriented North-South-East-West or, equivalently, to assume the plastic sheet is aligned with the table. Now imagine cutting the tablecloth into one-inch squares in an aligned grid pattern, pin one of the stained squares to the table, oriented as it was originally, and stack (without rotating any square) all other stained squares neatly on top of it.

The stain is now within one square, but since the area of the stain is less than one square inch (and can be reduced only through stacking), some squares remain stain-free. Now pick a stain-free point and place the plastic so its intersection points lie directly on the point.

Since all other intersection points are outside the stained square, no intersection point touches the stacked stain. But what if the tablecloth were sewn back together? Each stained square would then be translated by an integral number of tablecloth squares

East or West and North or South back to the original position. It would then bear the same relationship to the plastic sheet's grid points it did before; that is, it would miss them.

2. Covering Dots on a Table.

Solution. We had to show that any 10 dots on a table can be covered by non-overlapping \$1 coins, in a problem devised by Naoki Inaba and sent to me by his friend, Hirokazu Iwasawa, both puzzle mavens in Japan.

The key is to note that packing disks arranged in a honeycomb pattern cover more than 90% of the plane. But how do we know they do? A disk of radius one fits inside a regular hexagon made up of six equilateral triangles of altitude one. Since each such triangle has area $\sqrt{3}/3$, the hexagon itself has area $2\sqrt{3}$; since the hexagons tile the plane in a honeycomb pattern, the disks, each with area π , cover $\pi/(2\sqrt{3}) \sim .9069$ of the plane's surface.

It follows that if the disks are placed randomly on the plane, the probability that any particular point is covered is .9069. Therefore, if we randomly place lots of \$1 coins (borrowed) on the table in a hexagonal pattern, on average, 9.069 of our 10 points will be covered, meaning at least some of the time all 10 will be covered. (We need at most only 10 coins so give back the rest.)

What does it mean that the disks cover 90.69% of the infinite plane? The

easiest way to answer is to say, perhaps, that the percentage of any large square covered by the disks approaches this value as the square expands. What is “random” about the placement of the disks? One way to think it through is to fix any packing and any disk within it, then pick a point uniformly at random from the honeycomb hexagon containing the disk and move the disk so its center is at the chosen point.

3. Placing Coins.

Unsolved. The solution to Puzzle 2 doesn't tell us how to place the coins, only that there is a way to do it. Is there a constructive proof? Yes, and we can use the solution to Puzzle 1 (concerning the stain) to find it. I leave it to your imagination to follow up.

That proof can be used to increase the number of dots to 11 or 12, still using only an aligned hexagonal lattice of coins. However, since we aren't restricted to a lattice, it seems plausible that quite a few dots can be covered, perhaps as many as 25 (see the August column). If you figure out a dot pattern with, say, 30 or fewer points you think can't be covered by unit disks, please send to me, along with your reasoning.

All readers are encouraged to submit prospective puzzles for future columns to puzzled@cacm.acm.org.

Peter Winkler (puzzled@cacm.acm.org) is Professor of Mathematics and of Computer Science and Albert Bradley Third Century Professor in the Sciences at Dartmouth College, Hanover, NH.

Future Tense, one of the revolving features on this page, presents stories and essays from the intersection of computational science and technological speculation, their boundaries limited only by our ability to imagine what will and could be.

DOI:10.1145/1810891.1810918

Greg Bear

Future Tense Little Brother Is Watching

In a world of technology and fear, the public gets to know what it wants to know... and more than it can possibly digest.

A WET, BUSY highway at dusk. A pickup truck is tailgating you. Its front bumper pushes up to within six feet of your car. The driver glares, though he can easily pass on either side. You're in his way.

Your bumper camera is activated by a proximity sensor and snaps a high-res image of the truck and its license plate, then sends it to the highway patrol. Your car's equipment is law-enforcement certified, the evidence is clear, and the driver of the pickup is issued a ticket based on four such incidents within a 10-minute period. A record of aggressive driving can result in hefty fines and even a suspended license. Some municipalities offer the equipment for free to drivers in their jurisdictions. The impressive revenue stream can be useful in communities where angry voters no longer want to pay taxes.

These are often the same folks most interested in locking 'em up and throwing away the key—until they themselves are snagged and fined.

The whole city and most of the highway system will soon be monitored by millions or even billions of small, cheap, hi-res cameras, some embedded in paint and masonry—all feeding into servers and computer hubs that constantly process imagery, looking for suspicious or illegal behavior.

The computers pass along potential items of interest to tens of thousands of human contract operators employed at centers around the nation, where they compare possible "hits" to recent 911 calls and criminal databases and may pass them along to FBI and Homeland Security hubs, where they are further analyzed and refined.



One of the most important developments in surveillance will be facial-recognition software capable of comparing blurry videos taken from many different cameras and angles, as well as still photos. Computer enhancement will soon be able to approximate 3D scans from 2D sources. Still, even with improved software, fewer than one in 10 identification hits will be accurate. Computers still find it difficult to work with faces. Even many humans aren't very good at the art of comparison and identification.

A woman walks to her neighborhood market. A man follows her to the corner, dogging her every step. Built into the woman's cellphone is an aggression-warning monitor she triggers

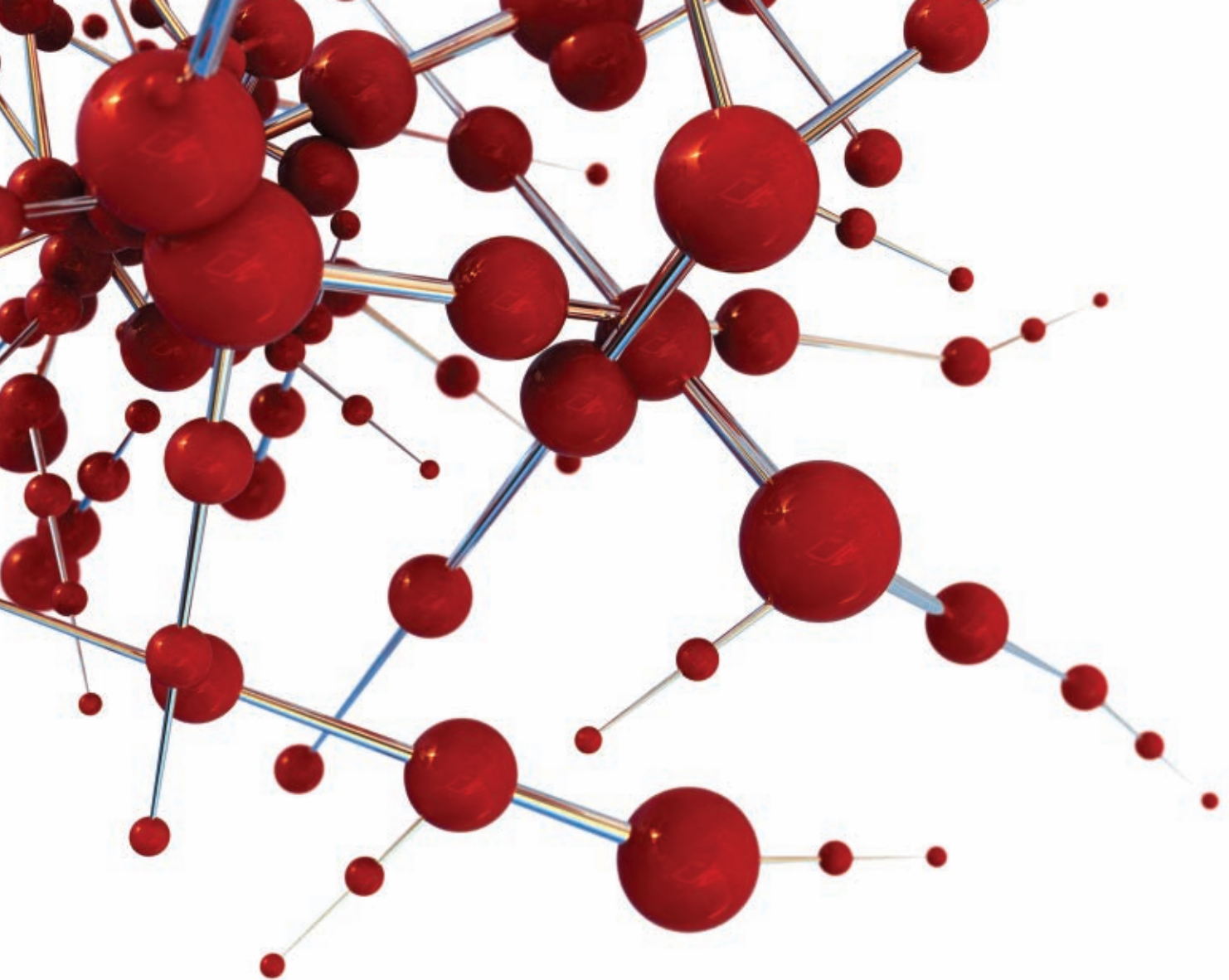
with a finger or a word. It activates a 4K video camera in her spex—what look like glasses and may in fact contain prescription lenses. As the man approaches, a video is recorded—surprisingly detailed, even in low light.

The video integrates streams from four cameras recording wide-angled images all around the woman.

The man's features are obscured by a hoodie. The woman's vest-embedded health sensor confirms that her heart-rate is up, she's starting to sweat, and the closeness and anonymity of the man indicates the likelihood of danger.

As the man in the hoodie comes within 30 feet of the woman, her computer sends an emergency alert to the local police department. CCTV cameras mounted throughout the neighborhood automatically backtrack, log, and process images from the last hour. A good view of the man's face from a single camera hundreds of yards away is compared to a database of known offenders. Seconds later, the woman is warned by a message flashed in her spex that the man is very possibly a convicted felon and sex offender. This automatically triggers a blaring alarm in the neighborhood.

The felon flees as neighbors come out on their porches to see what's up. Police drones the size of small birds flood the area, taking their own videos and guiding patrol cars. An arrest warrant is issued for the felon for parole violation. There's no place he can go without being tracked—unless he leaves the city completely. And even then, rural communi- [CONTINUED ON P. 111]



**CONNECT WITH OUR
COMMUNITY OF EXPERTS.**

www.reviews.com



Association for
Computing Machinery

Reviews.com

They'll help you find the best new books
and articles in computing.

Computing Reviews is a collaboration between the ACM and Reviews.com.

INTERACT 2011

13th IFIP TC13 Conference in Human-Computer Interaction

Lisbon, Portugal

September, 5-9 2011



www.interact2011.org

Honorary Chairs:

Don Norman, Larry Constantine,
Annelise Mark Pejtersen

General Conference Chairs:

Joaquim Jorge, Philippe Palanque

Full Research Papers Chairs:

Nicholas Graham, Nuno Nunes

Short Papers Chairs:

Daniel Gonçalves, Janet Wesson

Special Interest Group (SIGs) Chairs:

Gerrit van der Veer, Teresa Romão

Panels Chairs:

Regina Bernhaupt, Peter Forbrig,
Nuno Correia

Interactive Posters Chairs:

Monique Noirhomme-Fraiture,
Adérito Marcos

Demonstrations Chairs:

Verónica Orvalho, Greg Phillips

Doctoral Consortium Chairs:

Gitte Lindgaard, Manuel João Fonseca

Industrial Programme Chairs:

Antonio Câmara, Miguel Dias, Stacy
Hobson, Oscar Pastor, Virpi Roto

Workshops Chairs:

Julio Abascal, Nuno Guimarães

Tutorials Chairs:

José Creissac Campos, Paula Kotze

Student Design Competition Chairs:

Simone Diniz Junqueira Barbosa,
Luis Carriço

Organizational Overviews Chairs:

Teresa Chambel, Mary Czerwinski

Publicity Chairs:

Paula Alexandra Silva, Tiago Guerreiro

Keynote speakers co-Chairs:

John Karat, Jean Vanderdonckt

Student Volunteers co-Chairs:

Xavier Ferre, Effie Law

Publications co-Chairs:

Pedro Campos, Marco Winckler

Website Chairs:

Gerhard Weber, Alfredo Ferreira

Local Organization Chairs:

Alfredo Ferreira, Pauline Jepp

Conference Theme

The conference theme, Building Bridges, recognizes the interdisciplinary and intercultural spirit of Human-Computer Interaction (HCI) research. The conference welcomes research that bridges disciplines, cultures and societies. Within the broad umbrella of HCI, we seek high quality contributions opening new and emerging HCI disciplines, bridging cultural differences, and tackling important social problems. INTERACT 2011 provides a forum for practitioners and researchers to discuss all aspects of HCI, including these challenges.

Call for Submissions

INTERACT 2011 solicits submissions in a broad range of categories, addressing all aspects of Human-Computer Interaction. Submissions are welcome in the form of long and short research papers; workshop, tutorial, panel and special interest group proposals; industrial reports; interactive experience posters; organizational overviews, and applications to join the doctoral consortium and the student design competition.

Submission Categories

Submissions will be peer-reviewed by an international panel of experts and are invited in the following track categories:

- | | |
|---------------------------|------------------------------|
| ∴ Full Research Papers | ∴ Doctoral Consortium |
| ∴ Short Papers | ∴ Industrial Programme |
| ∴ Special Interest Groups | ∴ Workshops |
| ∴ Panels | ∴ Tutorials |
| ∴ Interactive posters | ∴ Student Design Competition |
| ∴ Demonstrations | ∴ Organizational Overviews |

Important Dates

Conference:

September 5th to 9th, 2011

Tutorial and Workshop Submission:

March 21th, 2011

Full Research Paper Submission:

January 10th, 2011 (abstract)

January 24th, 2011 (paper)

Submissions to Other Categories:

April 7th, 2011

Conference Language

INTERACT 2011 is an international conference whose official language is English. All submissions must be in English.

Follow us on:  tinyurl.com/facebook-interact2011

 twitter.com/interact2011



www.interact2011.org