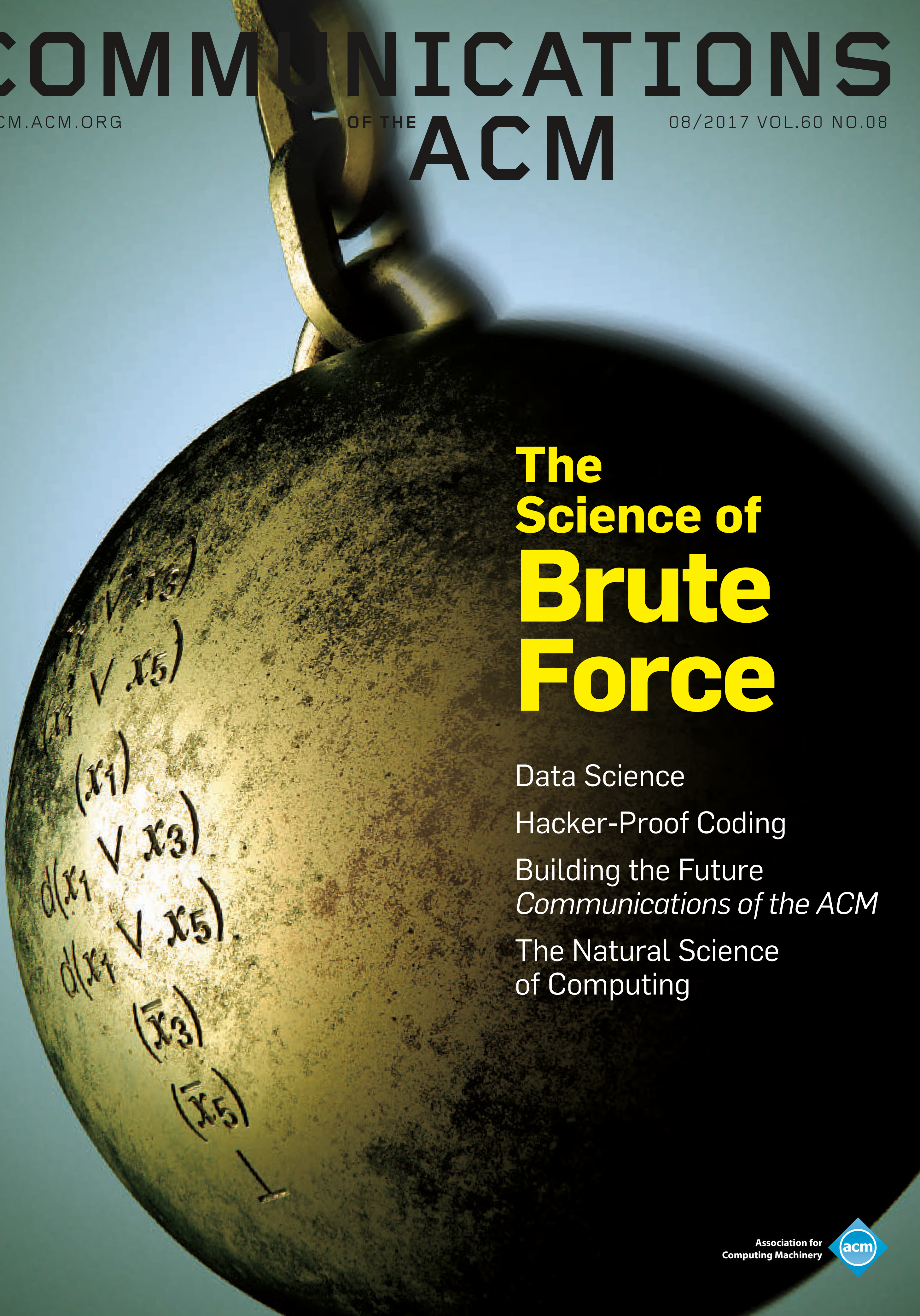
COMMUNICATIONS OF THE ACM

CACM.ACM.ORG

08/2017 VOL.60 NO.08



The Science of Brute Force

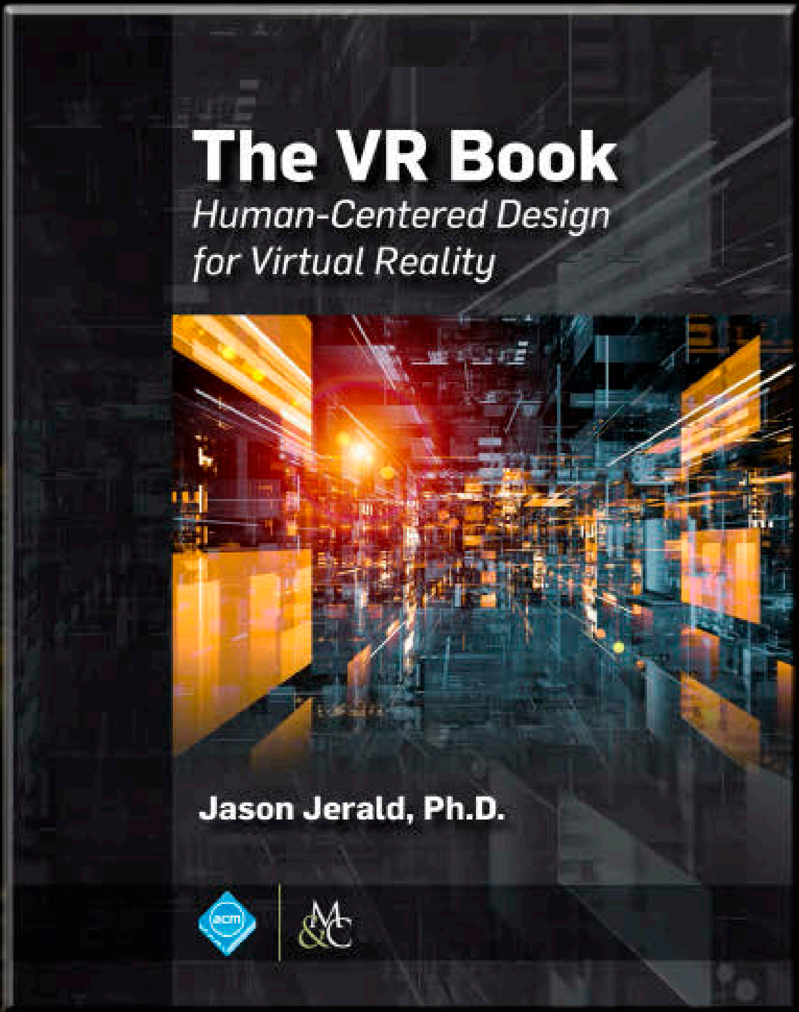
Data Science

Hacker-Proof Coding

Building the Future *Communications of the ACM*

The Natural Science of Computing

Association for Computing Machinery


The VR Book
Human-Centered Design for Virtual Reality
Jason Jerald, Ph.D.

SIGGRAPH
ASIA 2017
BANGKOK

CONFERENCE 27 – 30 November 2017
EXHIBITION 28 – 30 November 2017
BITEC, Bangkok, Thailand

10th Edition

THE CELEBRATION OF LIFE & TECHNOLOGY

CALL FOR SUBMISSIONS

Submit your works & be a presenter at SIGGRAPH Asia!

SIGGRAPH Asia 2017 invites you to submit your works and showcase your outstanding creative ideas and innovations at the 10th ACM SIGGRAPH Conference and Exhibition on Computer Graphics and Interactive Techniques in Asia, taking place from **27 – 30 November**, in Bangkok, Thailand.

Log-on to **sa2017.siggraph.org/submitters** to submit your works.

CONFERENCE PROGRAMS' SUBMISSION DEADLINES*:

DEADLINES	PROGRAMS
27 April 2017	Workshops' Proposals
23 May 2017	Technical Papers
30 May 2017	Emerging Technologies
1 June 2017	Art Gallery
13 June 2017	Symposium on Education
21 June 2017	Symposium on Mobile Graphics and Interactive Applications
28 June 2017	Courses
29 June 2017	Symposium on Visualization
15 July 2017	Student Volunteers - Team Leaders Application
19 July 2017	Computer Animation Festival
30 July 2017	VR Showcase
12 August 2017	Student Volunteers Application
15 August 2017	Posters Technical Briefs Workshops' Papers

**The submission time for all dates is 23:59 UTC/GMT*

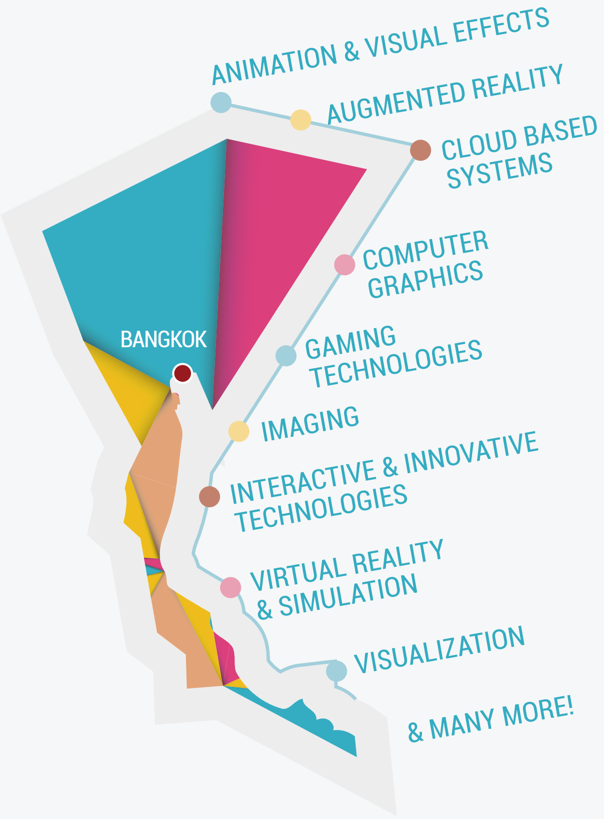
Visit sa2017.siggraph.org for more details.

CALL FOR EXHIBITORS & SPONSORS

Be a part of the SIGGRAPH Asia Exhibition – Asia's Digital Media Marketplace

Meet close to 7,000 technical and creative industry experts and individuals from over 60 countries and regions face-to-face to explore business opportunities, partnerships, and to strengthen existing relations – all in person at SIGGRAPH Asia 2017. Book your stand now to secure your preferred location.

Contact Clariss Chin at **+65 6500 6722** or **clariss.chin@siggraph.org** for more information on the exhibit space options and fees, as well as sponsorship packages.



ANIMATION & VISUAL EFFECTS
AUGMENTED REALITY
CLOUD BASED SYSTEMS
COMPUTER GRAPHICS
GAMING TECHNOLOGIES
IMAGING
INTERACTIVE & INNOVATIVE TECHNOLOGIES
VIRTUAL REALITY & SIMULATION
VISUALIZATION
& MANY MORE!

Sponsored by

SA2017.SIGGRAPH.ORG

Organized by koelnmesse
we energize your business | since 1924

COMMUNICATIONS OF THE ACM

Departments

5 Editor's Letter
Building the Future *Communications of the ACM*
By Andrew A. Chien

7 Cerf's Up
In Praise of Under-Specification?
By Vinton G. Cerf

8 Letters to the Editor
Embed Ethical Guidelines in Autonomous Weapons

10 BLOG@CACM
How Adults Ages 60+ Are Learning to Code
Philip Guo discusses his project studying older adults that have chosen to learn computer programming.

27 Calendar

100 Careers

Last Byte

104 Future Tense
Turing's Taxi
Ride with an autonomous AI cab driver that might actually know too much about where it's going ...
By Brian Clegg

News

12 **Hacker-Proof Coding**
Software verification helps find the faults, preventing hacks.
By Esther Shein

15 **Why Virtual Reality Will Transform a Workplace Near You**
A clutch of companies are changing how work gets done—by using virtual reality and augmented reality technologies.
By Logan Kugler

18 **AI in Contact Centers**
Artificial intelligence technologies are being deployed to improve the customer service experience.
By Keith Kirkpatrick

20 **Charles P. 'Chuck' Thacker: 1943–2017**
By Lawrence M. Fisher

Viewpoints

22 Historical Reflections
Prophets, Seers, and Pioneers
Reflections on historical prognostications for the future.
By David P. Anderson

26 Education
Is the U.S. Education System Ready for CS for All?
Insights from a recent Google-Gallup national research study seeking to better understand the context of K–12 CS education.
By Jennifer Wang

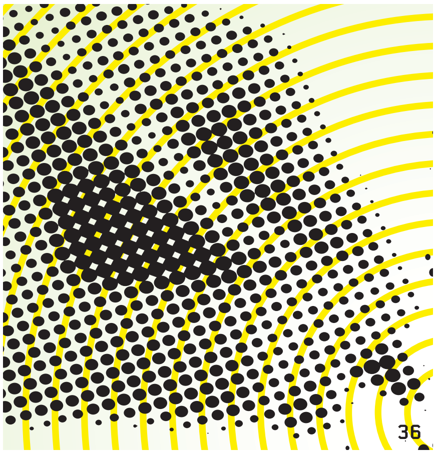
29 Kode Vicious
The Observer Effect
Finding the balance between zero and maximum.
By George V. Neville-Neil

31 Viewpoint
The Natural Science of Computing
As unconventional computing comes of age, we believe a revolution is needed in our view of computer science.
By Dominic Horsman, Vivien Kendon, and Susan Stepney

(L) PHOTO: ROBIN VAN LONKHUIJSEN/AFP/GETTY IMAGES; (R) IMAGE BY TONI V/SHUTTERSTOCK

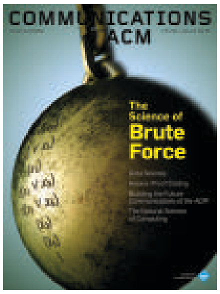
08/2017
VOL. 60 NO. 08

Practice



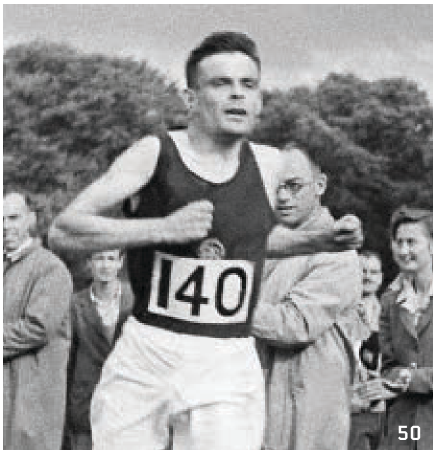
36 **Now That We Can Write *Simultaneously*, How Do We Use That to Our Advantage?**
Word processors now make it possible for many authors to work on the same document concurrently. But what can they actually do?
By Ricardo Olenewa, Gary M. Olson, Judith S. Olson, and Daniel M. Russell

44 **Small-Data Computing: Correct Calculator Arithmetic**
Rounding errors are usually avoidable, and sometimes we can afford to avoid them.
By Hans-J. Boehm



About the Cover: Brute force—an exhaustive search of all viable configurations of a problem—is one way to find a workable solution. But what would it take to trust the result of running complex algorithms on a multitude of computers over a long period of time? Marijn Heule and Oliver Kullmann explore the power of the proof beginning on p. 70. Cover illustration by Peter Crowther Associates.

Contributed Articles



50 **Turing's Pre-War Analog Computers: The Fatherhood of the Modern Computer Revisited**
Turing's machines of 1936 were a purely mathematical notion, not an exploration of possible blueprints for physical calculators.
By Leo Corry

Watch the authors discuss their work in this exclusive *Communications* video. https://cacm.acm.org/videos/turings-pre-war-analog-computers

59 **Data Science: Challenges and Directions**
While it may not be possible to build a data brain identical to a human, data science can still aspire to imaginative machine thinking.
By Longbing Cao

Review Articles

70 **The Science of Brute Force**
Mathematics solves problems by pen and paper. CS helps us to go far beyond that.
By Marijn J.H. Heule and Oliver Kullmann

Watch the authors discuss their work in this exclusive *Communications* video. https://cacm.acm.org/videos/the-science-of-brute-force

Research Highlights

82 Technical Perspective
Unexpected Connections
By Marc Shapiro

83 **The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors**
By Austin T. Clements, M. Frans Kaashoek, Eddie Kohler, Robert T. Morris, and Nickolai Zeldovich

91 Technical Perspective
Linking Form, Function, and Fabrication
By Helmut Pottmann

92 **Spin-It: Optimizing Moment of Inertia for Spinnable Objects**
By Moritz Bächer, Bernd Bickel, Emily Whiting, and Olga Sorkine-Hornung

(L) IMAGE COMPOSITION BY ANDRIJ BORYS ASSOCIATES, USING SHUTTERSTOCK; (R) PHOTO: NATIONAL PHYSICAL LABORATORY © CROWN COPYRIGHT/SCIENCE PHOTO LIBRARY

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.

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.

CONTACT POINTS
Copyright permission
permissions@hq.acm.org
Calendar items
calendar@cacm.acm.org
Change of address
acmhelp@acm.org
Letters to the Editor
letters@cacm.acm.org

WEBSITE
http://cacm.acm.org

AUTHOR GUIDELINES
http://cacm.acm.org/

ACM ADVERTISING DEPARTMENT
2 Penn Plaza, Suite 701, New York, NY
10121-0701
T (212) 626-0686
F (212) 869-0481

Advertising Sales Account Manager
Ilia Rodriguez
ilia.rodriguez@hq.acm.org

Media Kit acmmediasales@acm.org

Association for Computing Machinery (ACM)
2 Penn Plaza, Suite 701
New York, NY 10121-0701 USA
T (212) 869-7440; F (212) 869-0481

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 $269.

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) 626-0686.

Single Copies
Single copies of *Communications of the ACM* are available for purchase. Please contact acmhelp@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

Printed in the U.S.A.

Association for Computing Machinery

PLEASE RECYCLE THIS MAGAZINE

DOI:10.1145/3121002

Andrew A. Chien

Building the Future *Communications of the ACM*

Change is continual. All living things are changing—continual growth and renewal. A healthy bacteria colony regenerates hourly; Atlantic cod mature in 2 to 8 years, regenerating their shoals every few years. Even our bodies, seemingly static, create 225 billion new cells *every day*, replacing our 35 trillion cells every six months. Likewise, the *Communications of the ACM* team is dynamic; a collection of passionate leaders, making change and creating the future CACM through the actions, initiatives, and goals that we—the community of contributors, the editorial and production team, and the ACM membership—are pursuing today. In that vein, here are some of my ambitions for the Future CACM.

Building on Success and Excellence. CACM is excellent. Nearly every day, an ACM member tells me "it's the only publication I always read," and CACM's website and app provide easy access. Many members contribute articles to CACM as well. This is fantastic! But in 2017, the leading computing professional society, can and must do more.

First and foremost, we will maintain and grow CACM's excellence, continuing to highlight and disseminate compelling research, critical news, and incisive viewpoints. Central to this objective is CACM's position as a key leadership voice of the computing community. Leveraging computing's rise, CACM should become a broader and more influential voice. We will expand coverage and, as appropriate, create new features and venues that stimulate and lead the discourse where computing's progress and impact is most dramatic (for example, artificial intelligence, Internet of Things (IoT), and online issues from "radicalization," "fake news," "cybersecurity," and more). But beyond its core, CACM has several key dimensions of challenge and opportunity.

▸ How should CACM exploit new media to increase reach, access, and engagement? The millennial computing professionals were raised on smartphones, many where smartphones define Internet access. How can CACM reach and engage information "snackers" on handhelds? CACM must be engaging, and perhaps interactive, immersive, and more.

▸ Can CACM grow an energized, interactive community, creating a new venue for the voice of ACM members? (Showcasing and empowering their leadership, expertise, activities, and impact.) Our greatest unique strength is ACM members' deep and broad technical expertise. If CACM can give the leadership of individual members greater voice, ACM's impact and global relevance will increase around the world.

Grow the Computing Professional Community. ACM is the leading global society of computing professionals. Can CACM be relevant for *all* computing professionals across the globe? CACM's core strength today is excellence and breadth across research, practice, and policy; topics that reach across technology companies and academe. As "everyone codes" and information underpins every aspect of society, the professional community and its needs are growing rapidly. ACM membership should be indispensable for computing professionals working at the state of the art from academe to the tech industry worldwide. To reach this broad and expanding community, CACM must not only build on our successes, but also think and engage ambitiously!

ACM's membership today is strongest in North America and Europe, but it's clear that sustained global leadership must recognize the demographics of growing communities in Asia (China, India, and more), South America, and even the Middle East and Africa. How will we make CACM relevant in these varied settings? How can we give members in every geography a sense of ownership and participation? These significant challenges surely require creative engagement of leaders from all of these regions.

We need YOU!

These are my ambitions for CACM, but your ambitions can shape CACM too! Can we do more? Certainly. Are there other critical directions? Perhaps. How can they be achieved? ACM is a volunteer organization, so progress depends on your inspired and creative efforts. We need your help!

We need new members of the editorial board to drive ambitious new agendas, but also to sustain excellence. We need new member volunteers in diverse geographical, technical, and institutional settings to bring their creative perspectives, passion, and energy. They will create new CACM elements, and enable CACM to reach the broadest computing community. And we also need volunteers with vision and ambition to create a CACM that is not only a publication, but also gives members new venues and voice for their leadership and impact. Join us in building the future *Communications of the ACM*.

Email your ideas (eic@cacm.acm.org)! And, of course your offers to help!

Andrew A. Chien, EDITOR-IN-CHIEF

Andrew A. Chien is the William Eckhardt Distinguished Service Professor in the Department of Computer Science at the University of Chicago, Director of the CERES Center for Unstoppable Computing, and a Senior Scientist at Argonne National Laboratory.

ACM Europe Conference

Barcelona, Spain | 7 – 8 September 2017

The ACM Europe Conference, hosted in Barcelona by the Barcelona Supercomputing Center, aims to bring together computer scientists and practitioners interested in exascale high performance computing and cybersecurity.

The High Performance Computing track includes a panel discussion of top world experts in HPC to review progress and current plans for the worldwide roadmap toward exascale computing. The Cybersecurity track will review the latest trends in this very hot field. High-level European Commission officials and representatives of funding agencies are participating.

Keynote Talk by ACM 2012 Turing Award Laureate Silvio Micali,
"ALGORAND: A New Distributed Ledger"

Co-located events:

- ACM Europe Celebration of Women in Computing: WomENcourage 2017 (Requires registration, https://womencourage.acm.org/)
- EXCDI, the European Extreme Data & Computing Initiative (https://exdci.eu/)
- Eurolab-4-HPC (https://www.eurolab4hpc.eu/)
- HiPEAC, the European Network on High Performance and Embedded Architecture and Compilation (https://www.hipeac.net/)

Conference Chair: Mateo Valero, Director of the Barcelona Supercomputing Center

Registration to the ACM Europe Conference is free of charge for ACM members and attendees of the co-located events.

acm Europe Council

http://acmeurope-conference.acm.org

DOI:10.1145/3110531

Vinton G. Cerf

In Praise of Under-Specification?

Eric Schmidt, executive chairman of Alphabet (Google's parent company), recently drew my attention to the notion of "under-specification." He reminded me that the Internet had benefited strongly from this concept. Several specific examples came to mind. The Internet Protocol (IP) specification does not contain any information about routing. It specifies what packets look like as they emerge from or arrive at the hosts at the edge of the Internet, but routing is entirely outside of that specification[a] partly because it was not entirely clear what procedures would be used for Internet routing at the time the specification was developed and, indeed, a number of them have been developed over time. There is nothing in the specification that describes the underlying transmission technology nor is there anything in the specification that speaks to how the packet's payload (a string of bits) is to be interpreted. These matters are open to instantiation independent of the specification of packet formats.

Some of the under-specification can be a manifestation of *layering* that figured strongly in the ARPANET host-host protocols and was carried over in the Internet Protocol suite. The idea is that while there is a well-defined interface between the layers that specifies how information crosses the layer boundary, the details of the layer above or below are hidden. This feature allows for changes in the implementation of and even the characteristics of the upper or lower layer. For example, above the IP layer, one finds a number of different protocols such as User Datagram Protocol (UDP) or Transmission Control Protocol (TCP) or Real-Time Protocol (RTP) that all send and receive Internet packets but they use and interpret the IP packet payloads in different ways. Below the IP layer one finds a variety of different transmission technologies including Ethernet, Multi-protocol Label Switching, Frame Relay, Asynchronous Transfer Mode, Dense Wave-Length Division Multiplexing, and many others. The IP layer doesn't really care how the packets are transported.

What is interesting to contemplate is whether the notion of under-specification that induces flexibility and anticipates new but unknown developments can be codified in a concrete way beyond the purely conceptual. Is there a way to measure the degree of specification in the way that Claude Shannon found to specify information as entropy independent of semantics? Can something be fully specified, partly specified, or completely unspecified and how would these be described or measured more precisely? In circuit design, for instance, there is the notion of "don't care" for some values in a Boolean representation. Can this notion be applied to program specification as well as to protocol specifications? Are there design principles that one can derive from this notion of under-specification?

I am reminded of an anecdote told about doing business with Chinese manufacturers. American companies produced very detailed specifications of what was to be fabricated down to the last detail and the Chinese companies produced exactly what was asked, at a price. But a Chinese company produced a less specific specification, leaving room for the manufacturer to innovate, leading to a design that was less expensive, easier to manufacture, and to maintain.

One of my oldest friends, Jonathan Postel, was the Internet Assigned Numbers Authority for many years and was often quoted: "Be liberal in what you accept and conservative in what you send," in reference to the implementation of protocols. His dictum was aimed at improving interoperability. Of course, people who are particularly concerned about security might take issue with this particular nostrum (and some have!).

As may be apparent to readers who have gotten this far, I am not yet sure there is a there *there*, but I am fascinated by the possibility that it might be possible to extract some design principles from this notion that would lead to potentially more robust and adaptable designs. Think about what makes a chair a *chair*. It's a thing to sit on, has legs and usually a back and maybe some arms. But there are *so many* things we recognize as chairs that are quite varied in their specifics. Flexible design suggests to me that under-specification has something to do with *essence* or core concepts. I hope interested readers will take a moment to share their thoughts, particularly if they see more deeply into this idea than I have at the present.

Vinton G. Cerf is vice president and Chief Internet Evangelist at Google. He served as ACM president from 2012–2014.

[a] This is not precisely correct since the notion of "source routing" is part of the specification and allows a host to force packets to flow along a path specified by intermediate IP addresses, but the general route generation and selection process is independent of the IP specification itself.

DOI:10.1145/3117849

Embed Ethical Guidelines in Autonomous Weapons

As a combat veteran and more recently an industry technologist and university professor, I have observed with concern the increasing automation—and dehumanization—of warfare. Sarah Underwood's discussion of autonomous weapons in her news story "Potential and Peril" (June 2017) highlighting this trend also reminded me of the current effort to update the ACM Code of Ethics, which says nothing about the responsibilities of ACM members in defense industries building the software and hardware in weapons systems. Underwood said understanding the limitations, dangers, and potential of autonomous and other warfare technologies must be a priority for those designing such systems in order to minimize the "collateral damage" of civilian casualties and property/infrastructure destruction.

Defense technologists must be aware of and follow appropriate ethical guidelines for creating and managing automated weapons systems of any kind. Removing human control and moral reasoning from weapons will not make wars less likely or less harmful to humans.

Harry J. Foxwell, Fairfax, VA

When the Web Arrived for Me

Regarding Neil Savage's excellent news story on Tim Berners-Lee, "Weaving the Web," and Leah Hoffman's likewise excellent Q&A with Berners-Lee, "This Is for Everyone," (both June 2017), I would also add that the cost of computing has come down so much that organizations today are able (if willing) to allow outsiders to use their CPU resources. This would have been unthinkable in the mid-1980s. A computing colleague later suggested a routine based on the Bob Newhart-humor model, characterized by a phone call with a skeptical boss regarding new products or technologies. The audience would usually hear only the boss's side of the conversation; see, for example, the "Bob Newhart Tobacco video (Sir Walter Raleigh phone conversation)" at https://www.youtube.com/watch?v=_XDxAzVEbN4. Consider Tim Berners-Lee or other World Wide Web evangelist trying to convince his boss that letting the whole world tap the organization's expensive and secure CPU cycles to support the Web for the benefit of all humanity would indeed be a good idea. As with Newhart, we would imagine hearing it from the boss's perspective. For me, I knew Berners-Lee was onto something when only a month or so after I began experimenting with Mosaic and Netscape, I saw http://www.coca-cola.com printed on a can of Coke.

W. Terry Hardgrave,
Cross Junction, VA
ACM Member since 1967

How to Really Encourage Women in Computing

Reflecting on my involvement in the software profession in the 1950s and 1960s, when male and female participation were roughly equal, I can totally agree with Valarie Barr's view in her "From the ACM W-Chair" column "Gender Diversity in Computing: Are We Making Any Progress?" (Apr. 2017) and am astonished how few women enter or stay in the field 60-odd years later. Barr wrote that women are "hemorrhaging out the side and back doors" of the field after five years. The women I knew who headed out those doors did so because they became mothers, so I think it important to compare the female dropout rate among software professionals with similar dropout rates in other fields. I suspect, at least in earlier times, they would be similar, though I am less certain about those rates today. Nevertheless, this kind of data is vital for understanding what is happening. Also, Barr mentioned ACM financial support for

"Studies show engineering has the highest turnover rate for women compared to, say, accounting, law, and medicine, though the majority continues working."

female computer science students "to attend research conferences." But attending research conferences and the longevity of practitioner employment have almost nothing in common. Surely ACM's "encouragement" money could be spent in more direct and effective ways.

Robert L. Glass, Toowong, Australia

Author Responds:

There is no data about women's persistence in scientific and engineering fields, including computer science, in the 1950s and 1960s, and we cannot draw conclusions based on anecdotal information alone. Current studies show engineering today has the highest turnover rate for women when compared to, say, accounting, law, and medicine, though the vast majority continues working; only 22% who have left report they are now doing "family care." Other research shows engineering and science culture is a much more significant factor in women's lack of persistence than are family concerns. While attendance at research conferences may not translate directly into long-term practitioner employment, conference attendance does help sustain and increase excitement about staying in computer science.

Valerie Barr, South Hadley, MA

Causal Connections for Predictive AI

As part of the "ACM Panels in Print" section "Artificial Intelligence" (Feb. 2017), panelist David Blei said he believes computer science needs to identify the causal connections between data components, concluding that artificial intelligence, with its predictive capabilities, will be enhanced through causal inference. For example, the first step toward using AI in database systems is to analyze and create a data map of the complexity of the causal interconnections between the data components in a problem space. A data map connects a data component to other data components through causal interrelationships. A data map can be created through qualitative analysis of data collected for a particular problem. The qualitative analysis could then take the form of "thematic analysis,"[1] using systems diagramming to gain greater insight into the data. At that point it might be advantageous to start applying AI directly to the data.

Researching the complexity of database systems, I have thus created such a data map, which is now ready to move to the next stage of automation where predictive analytics can help improve management of database systems. Using an analogy of the CODEX, or Control of Data Expediently, my research into causal connections has identified a potential role for AI in automating continuously changing best practice, thus representing an agile approach to deciphering the complexity of interconnections and promising to help create an autonomous way to deliver best practice in database management.

Reference

1. Braun, V. and Clarke, V. Using thematic analysis in psychology. *Qualitative Research in Psychology 3*, 2 (2006), 77–101.

Victoria Holt, Bath, U.K.

Correction

In the ACM Member News column (May 2017), Dragomir Radev was mistakenly identified as a professor at the University of Michigan. Radev teaches at Yale University, where he also leads the Language, Information, and Learning at Yale (LILY) lab.

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

BLOG@CACM

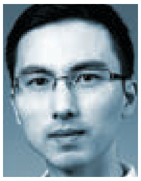
The *Communications* Web site, http://cacm.acm.org, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

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

DOI:10.1145/3105421 http://cacm.acm.org/blogs/blog-cacm

How Adults Ages 60+ Are Learning to Code

Philip Guo discusses his project studying older adults that have chosen to learn computer programming.



Philip Guo
Older Adults Learning Computer Programming: Motivations, Frustrations, and Design Opportunities
http://bit.ly/2rmGla5
May 15, 2017

I recently published and presented a paper at CHI 2017 (the annual ACM Conference on Human Factors in Computing Systems, https://chi2017.acm.org) called "Older Adults Learning Computer Programming: Motivations, Frustrations, and Design Opportunities" (http://bit.ly/2snS4LN). This paper won an Honorable Mention award at the conference. Here's a summary of the project.

There is now tremendous momentum behind initiatives to teach computer programming to a broad audience, yet many of these efforts (for example, Code.org, Scratch, ScratchJr, and Alice) target the youngest members of society: K–12 and college students. In contrast, I wanted to study the other end of the age spectrum: *how older adults aged 60 and over are now learning to code.*

Why study older adults in particular? Because this population is already significant and also quickly growing as we all (hopefully!) continue to live longer in the coming decades. The United Nations estimates that by 2030, 25% of North Americans and Europeans will be over 60 years old, and 16% of the worldwide population will be over 60. There has been extensive research on how older adults consume technology, and some studies of how they curate and produce digital content such as blogs and personal photo collections. But so far nobody has yet studied *how older adults learn to produce new technologies via computer programming*.

Thus, to discover older adults' motivations and frustrations when learning to code, I designed a 10-question online survey that asked about their employment status (such as working, semi-retired, retired), occupation, why they are learning, what resources they use to learn, and what has been the most frustrating part of their learning experience thus far.

The first challenge was finding a large-enough group of older adult learners to fill out my survey. Fortunately, I created a popular learn-to-code website called Python Tutor (pythontutor.com), which has gotten over 3.5 million total visitors from over 180 countries throughout the past decade. Approximately 16% of its user base self-report as aged 55 and older, so there are plenty of older adults learning to code on there.

I deployed my survey to the Python Tutor website from March 2015 to August 2016 and collected 504 responses. Respondents were, on average, 66.5 years old, and came from 52 different countries. Unsurprisingly, most were highly educated professionals in STEM (science, technology, engineering, mathematics) fields, since they are amongst the most tech-savvy of their generation. Specifically, 18% of respondents were (either current or retired) scientists and engineers, 18% were K–12 and college teachers, 12% were software developers hoping to learn new technologies, and 8% were business executives and managers.

Motivations

Why were our respondents learning

programming? The most common age-related motivations were:

- 22% wanted to learn to make up for missed opportunities during their youth.
- 19% wanted to keep their brains challenged, fresh, and sharp as they aged.
- 5% were directly motivated by younger relatives such as children or grandchildren.

Here's a great anecdote about learning to make up for missed opportunities during youth. A 67-year-old retired chief information officer wrote in his survey response:

> *"I did a little programming when I was in school, and when I first started working. However, I got "kicked upstairs" [into management] quite quickly, and was never able to program professionally. [...] I always wanted to be able to create programs but between work and family, never took the time. Now that I am retired, I am trying to fulfill the dream and learn."*

Relatedly, a 64-year-old retired networking engineer wrote about his desire to keep his brain sharp and to create technologies that benefit peers in his generation:

> *"First, by endlessly learning new things, I hope to delay or reduce the effects of senility on my brain. [...] Second, to take advantage of data produced by the many health-related, sensor-based monitors, I want to help myself and other senior citizens maintain an independent living lifestyle that is affordable by the masses."*

Frustrations

What got our respondents frustrated as they were learning to code? The three most commonly reported age-related frustrations were:

- 14% were frustrated by perceived cognitive impairments, such as memory loss and difficulty in concentrating.
- 11% were frustrated by lack of free time since they often had other duties, such as being a spousal caretaker.
- 10% were frustrated by lack of human contact with tutors or peers, since they must learn online and do not have convenient access to in-person classroom environments.

Older adults do not want to be patronized, to be talked down to, or made to play with "kids' toys."

A 71-year-old retired IT technician humorously wrote about his own perceived cognitive impairments:

> *"Given that I was a VERY early adopter of microprocessor/microcontroller technology, I have NO fear of the equipment or the concepts. But things that were "automatic" a few years back seem to take a lot more time and effort to digest and store than they used to. Early onset Alzheimer's? Probably not. ACS? (Advanced curmudgeon syndrome)—Probably some of that."*

Design Opportunities

Inspired by the findings from this study, I applied the Learner-Centered Design framework developed by Mark Guzdial in his book Learner-Centered Design of Computing Education: Research on Computing for Everyone (http://bit.ly/2seYahY) to propose design ideas for improving the learning experience for this older adult population. Three main themes emerged from my design process:

- ***Targeting:*** Like everyone else, older adults want to feel that programming curricula and tools "look like they're for me"—that is, that they are properly targeted to the motivations, needs, and aesthetic preferences of this population. They do not want to be patronized, to be talked down to, or made to play with "kids' toys." Several survey respondents mentioned brain training games (for example, from Lumosity) as being popular with their peers, so perhaps framing programming education in terms of brain-training games could work well for this audience.
- ***Contextualizing:*** It is also important to ground learning materials in contexts that engage this learner population, rather than trying to find a generic "one-size-fits-all" solution. Examples of relevant contexts here include structuring curricula around coding projects to help older adults curate digital media, to perform genealogical and historical storytelling, and to organize their personal healthcare data.
- ***Universal Design:*** The promise of universal design is that designing for the specific needs of a target population (such as older adults) can lead to designs that benefit *everyone*. In this case, we may want to design next-generation pedagogical programming environments that mitigate the effects of both cognitive and motor impairments, which will hopefully make it easier for older adults to learn to code without as many frustrations. If properly designed, these environments may actually end up benefiting learners of *all ages*.

Parting Thoughts

The tech world is notoriously youth-centered: popular conceptions of who learns and does programming are filled with images of young people, often under 30 years old. Also, age discrimination (see https://bloom.bg/2qCdIv6) is an all-too-common reality in the technology sector. To counteract these prevailing trends as people keep living longer in the coming decades, it is vital for older adults to have equal access to high-quality computing and programming education throughout their lives.

We have already made great strides in broadening participation of computing to traditionally underrepresented groups ... but there is still much, much more work to be done. Efforts to spread the power and joy of *computing for all* should also include people of all ages.

That's it for now! You can read my paper for more details: Older Adults Learning Computer Programming: Motivations, Frustrations, and Design Opportunities, at http://bit.ly/2snS4LN.

Philip Guo is an assistant professor of cognitive science at the University of California, San Diego.

Science | DOI:10.1145/3105423

Esther Shein

Hacker-Proof Coding

Software verification helps find the faults, preventing hacks.

At the University of Washington (UW) Medical Center, a radiotherapy system shoots high-powered radiation beams into the heads of patients, to treat cancers of the tongue and esophagus. Any software errors in the system could prove fatal, so engineers at the medical center have teamed with a group of computer scientists from the university to ensure the system will not fail, and that the beam will shut off if prescribed settings go out of tolerance.

This is made possible by a process known as software verification, and verifying implementations of critical systems like that radiotherapy setup is one of the things about which Zachary Tatlock is passionate. Over three years ago, Tatlock was a Ph.D. candidate giving a talk at the university on his thesis research in program verification. The lead engineer for the medical center's radiotherapy team was in the audience, and asked Tatlock how they could apply verification to that system. "That probably helped me get hired," Tatlock recalls. Today, he's an assistant professor of computer science at the university and, with other colleagues and students at UW, has also been working with the team at the medical center ever since.

LOGOS COURTESY OF DEEPSPEC.ORG

What makes the software verification process challenging in the case of the radiotherapy system described here is that it is written in a variety of languages, so different techniques have to be deployed to verify the software in its entirety. The system has about a dozen components, each with different levels of criticality.

Software for logging an event, for example, is not as critical as software that verifies the beam power has not become too high, Tatlock notes. "What we want to be able to do is ensure the reliability of all pieces,'' he says. "We want to make sure there are no bugs that can affect the parts that are critical." There are two or three components "where the rubber meets the road, and it's super-critical to get them right," he says.

The radiotherapy system team uses powerful verification methods ranging from automated theorem proving tools to manual proofs written by hand and checked by a proof assistant (a program that checks the correctness of proofs in expressive logic).

Preventing Software Hacks

Generally, computer code is written without any formal processes, and the main metric for testing it is simply trying it out and seeing whether it or not works. Testing does not necessarily guarantee all the bases have been covered that might occur at runtime, or that it would prevent a malicious attacker who reads the program from devising something clever with which to undermine it. Formal software verification relies on mathematical theorems and reasoning and uses deductive techniques to check the most critical aspects of a system. Proponents say this technique is making hacker-proof software possible.

"A lot of the ways attackers take over programs on machines and the Internet is by exploiting security vulnerabilities, and there are many kinds, such as the buffer overrun vulnerability, to which safe languages are just immune,'' notes Andrew Appel, professor of computer science at Princeton University, who is considered an expert in the program verification field.

Formal software verification uses methods that don't rely on running the program; rather, they analyze program text to prove things about its behavior on any possible input. Even using so-called "safe" languages such as Java to write programs doesn't necessarily guarantee they are correct, says Appel; they can still have bugs and do wrong things, just nothing catastrophic. Safety is always important, but correctness is crucial when it comes to the critical infrastructure components of a system, he emphasizes.

Safety has been proven to be an issue with critical systems before, and in at least one case, put patients' lives at risk. In 1982, Atomic Energy of Canada Limited (AECL) produced the Therac-25 radiation therapy machine. The system was involved in at least six accidents between 1985 and 1987, in which patients were given massive overdoses of radiation—sometimes as much as hundreds of times greater than normal, resulting in death or serious injury.

Among the findings of a commission that investigated the Therac-25 was that AECL did not have the software code independently reviewed, nor did it ever test the system's software and hardware until it was assembled at the hospital.

"So if you want your programs to be correct and not just safe, then you need to prove that your program behaves according to some specification,'' Appel says. "You have to write down in a formal way what would be correct in terms of how its output relates to its input, and then you have to find a way to assure that for any possible input, your implementation will satisfy its specification."

Appel is a member of a research project called DeepSpec, whose mission is to examine the full functional correctness of software applications and hardware so programs run the way they are supposed to run. To do this, DeepSpec is building tools for verifying that programs conform to deep specifications—granular, precise descriptions of how software behaves based on formal logic and mathematics—and that software components such as OS kernels provably conform to their deep specifications.

Another DeepSpec member, Yale University computer science professor Zhong Shao, along with a team of researchers there, wrote an operating system called CertiKOS which uses formal verification to ensure the code behaves exactly as is intended. "If the code and spec do not match, there is a bug or loophole,'' explains Shao. "What a certified operating system provides you with is a guarantee there are no such loopholes; under every situation, the code will behave exactly like its specification." This guarantees hackers cannot exploit the operating system, he says.

In addition to verification, the other factor that sets CertiKOS apart from other OSs is that it has an extensible operating system kernel, meaning it can easily be customized and adapted for many different settings so more features can be added, Shao says. CertiKOS also can run multiple threads (small sequences of programmed instructions) simultaneously on multiple central processing unit (CPU) cores, a process known as concurrency.

The major questions facing CertiKOS and other examples have to do with "semantics engineering," the process of defining specifications and proof methodologies to minimize the cost of revalidation, observes Suresh Jagannathan, a computer science professor at Purdue University.

DeepSpec is building tools for verifying programs, and software components such as OS kernels, conform to deep specifications.

"I don't consider this a limitation of these systems as much as an aspect of formal verification that will likely be increasingly important as we gain more experience with verification tools and proof assistants, and as we achieve more success in using these tools for verifying realistic systems," he says. It will be critical to determine what processes and methodologies need to be adopted to make proofs robust as specifications and implementations evolve, Jagannathan adds.

DeepSpec vs. Other Principles

The UW team uses DeepSpec principles to check the more heavy-duty components of the radiotherapy system. To assess the parts of the system that are not as critical, the team uses "lighter-weight, less-powerful techniques to ensure the correctness, so the guarantees for those parts aren't as strong, but it's a better engineering trade-off," Tatlock says. That's because the DeepSpec principles typically require highly trained humans to prove they function correctly, he says. "They take a lot of effort in that style, but you're rewarded by having a much stronger guarantee.''

This begs the question: if the DeepSpec techniques could make absolute, iron-clad guarantees of the verification of software, why don't we use them all the time to avoid crashes and bugs in all types of systems? The reason, says Tatlock, is cost. The proof system used by DeepSpec and the UW medical center radiotherapy team for the most critical components is too expensive to apply to all components, because it requires an expert to sit at a computer and type out proofs so the computer can check them for every version of the system.

This can be time-consuming and inefficient. "Ideally, if you proved one version and made a small change, you'd only make a small change to the proof, but that's not the way it works,'' Tatlock says. Whenever even a small change is made to software code, "it can have very large consequences,'' resulting in a big change in the proof. It might change some fact that's relied upon throughout the rest of the proof, he adds.

Other techniques are less powerful, like bounded model checking, which is a form of exhaustive testing, Tatlock says. This involves considering some component and showing every possible execution of steps up to some bound is correct. "So I might make sure for the logging system, there's no execution within 100 steps that ever crashes. I get that by automatically testing every execution up to 100 steps." Yet, he reiterates, the guarantee isn't as strong, and bounded model checking up to 100 steps does not tell you anything about the 101st step.

Tatlock and his colleagues have built a suite of tools the engineers use in their regular development process. They include a checker that allows them to formally describe the entire radiotherapy system to a computer and ensure the key components are individually correct. The researchers are now working on building verified replacements for those parts of the system. When all those individual system components are put together, he says, essentially, they are assured top-level safety verification.

The radiotherapy system is checked daily because "we want to make sure the code written by the engineers on that team will correctly turn off the beam if anything goes wrong,'' Tatlock says. The work is similar to DeepSpec's; it just emphasizes a different degree of automation.

While CertiKOS prevents one app from incorrectly reading the memory of another application, the UW team does not use it because CertiKOS is a traditional Unix OS meant for running Unix-style apps, and the preponderance of components in the radiotherapy system are embedded systems and just run code directly on the hardware, he says.

Like Tatlock, Appel and Shao stress that only certain types of critical software need to be verified. Already, Appel points out, some verification tools are commercially viable. For example, an optimizing C compiler in France called CompCert is being evaluated by Airbus and European certification agencies for use in compiling the fly-by-wire software used to fly the Airbus jetliner.

"Compared with the compiler Airbus currently uses, CompCert has the advantage of being proved correct, no matter what program is compiled," he says.

Other aeronautics agencies are also starting to use them. Formal verification tools "have been shown to be effective at finding defects in safety-critical digital systems including avionics systems," according to a 2017 report released by the U.S. National Aeronautics and Space Administration (NASA) Langley Research Center. Also, the U.S. Federal Aviation Administration (FAA) has issued guidelines that allow the use of formal methods tools to replace some of the methods it uses for "certification credit."

NASA acknowledges this will be a slow process, noting, "There are still many issues that must be addressed before formal verification tools can be injected into the design process for digital avionics systems." The report points out, "Most developers of avionics systems are unfamiliar with even the basic notions of formal verification, much less which tools are most appropriate for different problem domains," and skills and expertise will be needed to master the tools effectively.

The U.S. Defense Advanced Research Projects Agency (DARPA) has developed a program called High-Assurance Cyber Military Systems that takes a "clean-slate, formal methods-based approach to enable semi-automated code synthesis from executable, formal specifications." The approach ensures a hacker-proof system and "illustrates that we're now at a point where such systems can be deployed in truly mission-critical environments,'' notes Jagannathan.

In recent years, the auto industry has become aware of how vulnerable cars are to hacking, and "they are eagerly looking for solutions from security researchers and verification researchers," Appel says. He believes in the next decade there will be more industries using software verification, and more software available for purchase that has been verified.

For software verification to become widespread, however, there must be trusted compilers to translate these formal software verification methods, since they are written in high-level languages. Progress is being made, however, and when a trusted compiler becomes widely available, the issue of hacker-proof software may no longer be a wistful notion, but a concrete reality. C

Further Reading

Shao, Z.
Certified Software, *Communications of the ACM*, 53(12), pages 56-66, December 2010

Serna-M. E., and Morales-V. D.
State of the Art in the Research of Formal Verification, *Ingeniería, Investigación y Tecnología*, Oct.-Dec. 2014, pgs. 615-623. Volume 15, Issue 4.

D'Silva, V., Kroening, D., and Weissenbacher, G.
A Survey of Automated Techniques for Formal Software Verification, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, July 2008, Vol. 27, No. 7.

Souyris, J., Wiels, V., Delmas, D., and Delseny, H.
Formal Verification of Avionics Software Products, *FM 2009: Formal Methods*, Second World Congress, Eindhoven, The Netherlands, Nov. 2-6, 2009, Proceedings, pp. 532-546

Hoare, T.
The Verifying Compiler: A Grand Challenge for Computing Research. In: Hedin G. (eds) *Compiler Construction*. CC 2003. Lecture Notes in Computer Science, 2003, vol 2622. Springer, Berlin, Heidelberg

Esther Shein is a freelance technology and business writer based in the Boston area.

Technology | DOI:10.1145/3105444

Logan Kugler

Why Virtual Reality Will Transform a Workplace Near You

A clutch of companies are changing how work gets done—by using virtual reality and augmented reality technologies.

Collaboration is in. From offices with open floor plans to new apps that promise to reinvent messaging and collaboration, companies find it buzz-worthy and attractive to the bottom line when their teams work better together.

Need proof? Messaging app Slack has a $3.8-billion valuation. In January, collaboration app Trello was sold for $425 million to software company Atlassian. Tech heavyweight Amazon in February dropped Chime, a Skype and GoToMeeting competitor, seeing a potential profit center in the already crowded collaboration market.

Yet in the next few years, none of these companies may matter at all. That is a bold claim, but one that will come true sooner than anticipated if champions of virtual reality (VR) and augmented reality (AR) technology have their way.

VR places users in a virtual world, typically via a headset that immerses them fully in a digital environment. AR, however, lays digital visuals over what you see in the real world. Neither is new, but only recently have the technologies become affordable—and powerful enough—for adoption by consumers and businesses, thanks to advanced VR systems like the Oculus Rift.

The Rift headset immerses the user in a digital world to a degree unseen in the VR tech of the 1990s and early 2000s. Oculus' quality of VR was so impressive, the company was bought by Facebook for north of $2 billion in 2014.

At the other end of the spectrum, simple but popular apps like the smash-hit Pokemon Go put AR in the hands of anyone with a smartphone. In Pokemon Go, users hunt cartoon monsters in the real world, walking around their communities and interacting with digital content laid over real images captured by their smartphone's camera. While the app does not match Oculus' degree of realism, it became wildly popular nonetheless, breaking Apple's App Store record for most downloads in a week.

Microsoft's Hololens in use in a warehouse facility in The Netherlands.

PHOTO BY ROBIN VAN LONKHUIJSEN/AFP/GETTY IMAGES

Facebook's investment in Oculus jolted companies into action, sending a market signal that VR was here to stay. Pokemon Go's popularity proved that users were comfortable with—and even keen on—AR experiences. Companies like Microsoft, HTC, and Sony have acknowledged VR and AR's potential, too; each company has its own consumer-facing VR and AR hardware.

Today, Oculus' Rift headset and Touch hand sensors retail for just under $600. AR apps abound on smartphones. Solutions like Sony's PlayStation VR and Samsung's Gear VR headsets have entered the market. More consumers—and developers—are discovering the power of VR and AR.

Except it's no longer just for fun and games; thanks to relatively cheap and better technology, VR and AR are poised to transform how we *work*.

VR and AR companies see the technology as the natural evolution of popular collaboration apps; instead of desktop interfaces or pinging phone apps, however, they envision a world in which we collaborate with coworkers and peers in highly realistic virtual or virtually augmented environments—a more immersive, versatile, and natural way to get business done.

This is no pipe dream like the VR of decades past. Thanks to the commercial viability of popular systems and apps, ambitious VR/AR products are already changing how professionals work, train, and cooperate. Company reps use the technologies to better train for customer-service interactions and to troubleshoot issues faster in real time using digital models. Manufacturers rely on the technology to better collaborate on the design and maintenance of components, and hospital systems employ VR and AR to remotely train doctors faster, less expensively, and more effectively.

This is just the beginning. Your next meeting might take place in a VR environment that makes it easier to identify who's speaking and how others feel about your ideas (if Peter Diamandis' company High Fidelity, which creates "open source software for creating, hosting, and exploring shared VR experiences," has its way). Entire collaboration exercises or corporate retreats might be hosted in virtual environments (courtesy of technology like that offered by AltspaceVR, a virtual reality software company building a new communication platform already used by people in more than 150 countries) Training on everything from operating equipment to combatting sexual harassment at work could become a lot easier and more effective, thanks to VR in the workplace.

"Phones will disappear," predicts Ajay Shah, cofounder and business development head at Dotty Digital, maker of the first collaborative environment for working on three-dimensional (3D) models over a Web browser, a system which includes AR components. "Everything is moving to wearable tech and everything will be viewed with AR, so you can still capture your true environment. Everything will be synced from voice commands and eye tracking with potentially a watch for touch. The tech already exists."

That tech could make your workplace very different from the one you know today.

Real Business—Virtually

Dotty's AR solutions work with smartglasses from ODG, which raised the largest Series A (first significant round of venture capital financing) in wearables when it scored $58 million in 2016. Thanks to Dotty's tech, users see 3D models projected on their lenses; they can then collaborate digitally with other users viewing the same model or visuals, completely hands-free. Right now, Dotty sees the biggest use cases in retail, oil and gas, and manufacturing workplaces.

Such technology can change how retailers work and consumers shop. As an example, the company's 3D scanning app could be used by a sunglasses retailer to scan customer faces and digitally overlay different styles of shades. That fundamentally alters how service reps do their jobs; instead of answering sizing or availability questions, reps can provide higher-level consultations that require more training, but also increase the perceived value of the product or service they are selling.

Collaborative 3D models could change how manufacturers work, too. Instead of a field engineer constantly traveling between locations, she could troubleshoot machinery and refine product designs in real time, in tandem with colleagues back at the home office, from anywhere in the world using Dotty's collaborative AR environment. In turn, she could scale herself far more effectively, changing how her colleagues do their jobs and how her company hires.

Thanks to the commercial viability of popular systems and apps, ambitious VR/AR products are already changing how professionals work, train, and cooperate.

This becomes even more compelling when the sensors and smart machinery that comprise the Internet of Things (IoT) enter the picture. The company's DottyView product works with Autodesk CAD software to visualize assets in 3D, along with live data from that equipment or vehicle's sensors. For instance, a ship captain could view a 3D model of his vessel complete with real-time status reports on various components. From there, problems could be diagnosed and solved from thousands of miles away. In an industry like commercial shipping, this reduces—or eliminates completely—the need for costly, time-consuming ship inspections that take place in all types of weather and conditions, including dangerous ones.

"These types of AR solutions allow remote workers to coordinate in real time on complex models and assess solutions much quicker for field teams," says Shah.

Workers Without an Office

AR and VR in the workplace can be life-changing, especially when your workplace is a hospital tasked with saving lives. Osso VR is a surgical simulation platform that trains residents and helps veteran surgeons warm up for procedures in a highly realistic virtual environment.

"During my surgical training, I noticed that healthcare providers mainly learned their technical skills on the job, often taking the expression 'see one, do one, teach one' quite literally," says Justin Barad, cofounder of the company and a board-certified orthopedic surgeon. "This led to frequent situations where teams didn't seem to be as proficient as they could be during certain procedures because of a lack of knowledge and experience."

Users practice specific procedures using Osso VR and a set of controllers that track hand movements. That speeds up how surgeons go about their work.

"If I want to learn about a new technique or device, I typically will attend a course," explains Barad. "These are usually weekend affairs in a remote location, sometimes even across the country. Finding the time to make it to one of these courses is difficult as surgeons barely have enough time outside of the hospital as it is."

With a system like Osso VR, hospitals and healthcare systems can ensure these professionals are completely prepared and confident before they perform life-and-death procedures, reducing the likelihood of mistakes and improving outcomes—all with a fraction of the time and hassle required by in-person training.

Yet AR and VR in the workplace do not need to save lives to improve them. Atheer creates AR/VR hardware and software solutions for deskless employees. The company's AiR Suite provides visual and non-interruptive collaboration, communication, and workflow management on commercially available smartglasses. The result? Workers can collaborate with headquarters without taking their hands, or eyes, off the job in front of them.

The company also manufactures its own smartglasses that are compatible with the system, the AiR Glasses. Powered by the Android operating system, the glasses connect to Wi-Fi, Bluetooth, and 4G LTE for maximum access to digital information in the field. Workers control applications displayed on the glasses using hand gestures, head movements, and voice commands.

The company cites applications in logistics/warehousing, construction, and industrial sectors as target sectors for the technology. These industries share commonalities: field workers who need to learn and communicate, but who may not have the ability to use a mobile device or onsite machine to do so.

Then, of course, there are the companies who want to be the new Slack (that is, a popular collaboration solution), but for VR.

Software from AltspaceVR gives companies and individuals the ability to connect with others in a shared digital environment. Using a VR headset like the Oculus Rift, HTC Vive, or Samsung Gear VR, users can visually brainstorm like they are in the same room, or conduct more natural meetings than otherwise possible through video or voice conferencing.

A ship captain could view a 3D model of his vessel with real-time status reports on various components. From there, problems could be diagnosed and solved from thousands of miles away.

VR company High Fidelity also has a platform-first approach. The company provides users with an open-source system that works with major commercial VR headsets, which allows them to create highly scalable virtual environments using common tools.

These virtual environments offer real benefits to companies—especially as many firms implement remote work policies.

"A good portion of our team is remote and we're already seeing VR become useful as a productivity tool," says Barad at Osso VR. "We often have our daily meetings in VR. This allows us to function like we're all in the same physical space.

"I can see this supplanting videoconferencing in the near future."

Further Reading

Myers, B.
Going to Work in VR Will Actually Be Pretty Great—We Swear, *WIRED*, May 7, 2016, https://www.wired.com/2016/05/going-work-vr-will-actually-pretty-great-swear/

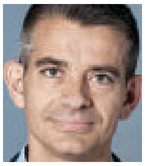
Lopez, M.
Augmented And Virtual Reality Fuel The Future Workplace, *Forbes*, December 11, 2016, http://www.forbes.com/sites/maribellopez/2016/11/11/augmented-and-virtual-reality-fuel-the-future-workplace/#68575e5c198e

McGhee, B.
How VR Will Change the Workplace, *AndroidPit*, February 9, 2017, https://www.androidpit.com/how-vr-will-change-the-workplace

Logan Kugler is a freelance technology writer based in Tampa, FL. He has written for over 60 major publications.

ACM Member News

PURSUING ENERGY EFFICIENT COMPUTING



"I am excited about making computing as energy efficient as possible, which means affordable and accessible," says Luca Benini, professor of Digital Circuits and Systems in the Department of Information Technology and Electrical Engineering at ETH Zurich. "This is what my research has focused on my entire career: having energy efficient computing at your fingertips whenever you need it—at a minimal cost and with high availability."

Benini earned his undergraduate degree in electrical engineering from the Università di Bologna, and received both his master's and Ph.D. degrees in that discipline from Stanford University.

Watching people at Stanford learn about things and then bring them into being struck a chord with Benini. "Science is not only learning, but making things practical and changing peoples' lives," he explains. "This was an attribute I first saw at Stanford and it changed my approach to problems and made me excited by what I was doing."

After graduating from Stanford, Benini started working at Hewlett Packard. Within a year, he had received an offer from his alma mater to become a professor of electrical engineering, which he accepted. While there, he also served as a visiting professor, first at Stanford and then at EFPL in Switzerland.

Benini has also worked as a consultant for industry.

He joined ETH Zurich in 2012. "I wanted to move from abstract research into making chips, which what I have been doing the past four years," he says.

Benini's focus remains on making computation increasingly energy efficient.

—*John Delaney*

Society | DOI:10.1145/3105442

Keith Kirkpatrick

AI in Contact Centers

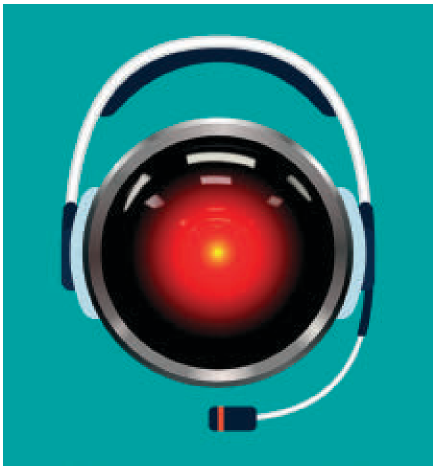
Artificial intelligence technologies are being deployed to improve the customer service experience.

Customer contact centers are most efficient when they are able to automate routine tasks and quickly route callers to human agents who can solve issues in a timely and courteous fashion. In years past, rules-based decision matrices (such as "press 1 for sales, press 2 for technical support") were the de facto standard for "intelligent" customer service systems, and often left customers frustrated and angry by the time they reached a live human being.

Advances in artificial intelligence are yielding significant benefits for organizations that deploy the technology in their call centers. Indeed, rather than simply being used to replace contact center workers, artificial intelligence (AI)-based technologies, including machine learning, natural language processing, and even sentiment analysis, are being strategically deployed to improve the overall customer experience by providing functionality that would be too time-consuming or expensive to do manually.

"It's a lot more prevalent than people think," explains Justin Robbins, content director for the International Customer Management Institute and HDI. He cites the example of cross-language email support that is made more seamless by the integration of natural language processing.

"You may be speaking German, but when it comes to me as an agent, I read it in English that is fully contextual, and when I reply in English, thanks to natural language processing, it is returned to you in German, and it's back in your natural language, in context," Robbins says, discussing the technology that is currently available and deployed in some international contact centers via their text-based chat applications. "To the customer, it has no impact on your experience, but we as agents, it now allows me to help you without the frustration of language barriers."



Adoption Driven by Customer Expectations

Perhaps the biggest driver of AI in customer contact centers is the consumer acceptance of AI technology in devices such as Apple's Siri, Amazon's Alexa, and Google's Home Assistant, which have conditioned consumers to be able to ask a question in a conversational, natural tone, and have the answer returned to them quickly. That has conditioned consumers to demand the same level of interaction when dealing with the customer care division of the companies and brands with which they interact on a regular basis.

"There's a set of evolutions in consumer behavior," says Alex George, chief technology officer of Astute Solutions, a provider of call center technology to businesses such as McDonald's Corp., British Airways, L'Oreal, and Dunkin' Donuts. "We see that people are trying to achieve more in short conversations."

Astute Solutions uses artificial intelligence technology in a few distinct ways. The company uses AI "bots" to handle routine tasks by utilizing natural language processing to interpret what customers are asking, search the business knowledge base system for an answer, and then interpreting this raw data into an intelligent, human-friendly response.

For example, according to George, "80% of the calls an airline receives to change a ticket do not result in the ticket changing," because the person may not efficiently be made aware of all of the terms and conditions involved, such as change fees or scheduling issues. The AI system can quickly provide this pertinent and relevant information to the caller, without engaging the services of a live agent. "By using bots, customer call volume can be reduced [significantly]."

Indeed, AI is being used by customer contact centers as a contextual knowledge management system. "Some of the work we do is around specifically making sure that customers and contact center agents have the right information in real time," George says, which allows them to serve customers with the information they need immediately, eliminating the need to escalate an issue to a specialist or manager.

Astute Solutions also uses machine learning to track the behavior of its agents, so the system can learn the most appropriate responses to questions, for use in its automated bots and to train other agents. The company using the system can specify thresholds for what constitutes a successful behavior—such as requiring that 80% of agents must recommend a specific course of action in order for the system to recognize that action as the "ideal" response, and can set the appropriate learning period, reviewing interactions over the past day, week, month, or other time frame.

Multi-Channel Support

Artificial intelligence can do more than simply recognize patterns in call center interactions. Andrew Burgess, a strategic advisor at Celaton, a U.K.-based provider of machine learning technology to contact centers, says that when deployed correctly, AI can provide enhanced service across a variety of platforms, which is how today's customers demand to engage with companies.

HEADSET IMAGE BY DACIAN G; HAL 9000 IMAGE BY CRYTERIA/CC-BY-3.0

For example, Celaton's inSTREAM platform allows many routine information requests, which often come in via email, to be automatically handled by an intelligent 'bot' that assesses the nature of the request, and then routes the query to the proper second-level live agent, rather than relying on costly front-line service agents.

"The best example we have at the moment is with one of our clients, a train operating company," Burgess says. "A lot of people email them, and they may be complimenting, complaining, or querying something. For example, the customer could be emailing in because they were on the 8:56 from London to Manchester and the Wi-Fi was out."

Burgess highlights the power of machine learning and natural language processing to quickly process front-end requests, which often make up a significant amount of call volume and labor costs. "inSTREAM essentially reads that email, understands what the customer is asking, and then will categorize that email and send it to the right person in the organization," he says.

"It's really taking that front-end input, understanding what the problem is, and then finding the best person to [handle] it," Burgess says. The technology has "reduced the requirement for labor by 85% on that task."

AI technologies such as machine learning can also be used to help customer service agents in their real-time phone interactions with customers. One of the ways AI has been used is to monitor and analyze speech patterns and inflections of callers, as well as reviewing specific words, to determine when an interaction may be in danger of escalating. Indeed, "sentiment analysis," where a system will detect changes in tone, speech patterns, or volume, can often be useful not only in addressing a situation in real time that needs to be escalated to a manager, it can also be used as a training aid so agents can learn to better recognize signs of stress or anger during an interaction. It can also suggest ways for an agent to reduce the stress level of a conversation.

Challenges with AI

Despite the ability of AI to improve customer experiences, many call centers and organizations have not yet adopted the technology.

One issue slowing the proliferation of AI in customer contact centers is "the work and effort that's required to program on the back end … it's not fine-tuned out of the box."

"Cost is the upfront issue," Robbins says, noting the initial implementation of new technology can be a barrier for an organization that has already made substantial investments into existing technology and agents. However, "the hidden one, which [companies that want to deploy AI-based customer service] don't realize until they start talking to people, is the work and effort that's required to program on the back end. That's the other thing about analytics and AI—it's not fine-tuned out of the box."

The sheer number of possible phrases, words, and interactions does make it more challenging to automate the customer service experience, though with machine learning technology that can review thousands or millions of interactions, organizations can tailor responses based on its learnings.

"The other area where inSTREAM has additional capability is it will suggest possible answers," Burgess says, noting that Celaton's machine learning technology can review thousands of possible answers to a particular question, and then filter and return three or four choices that best address the question.

"It makes the whole process much more efficient," Burgess says. "It's the ability to extract meaningful, structured data from unstructured input," like text or email messages sent by customers.

Nevertheless, ICMI's Robbins notes most of the systems currently available—even those that feature some degree of machine learning—still require a significant amount of training and programming to incorporate organization-specific terminology and processes. Even similar types of companies may not use the same words or phrases to refer to similar tasks, and it's up to the organization to customize the system for their needs, which can be a costly and time-consuming process.

"When [customer service] is done well, you still need live agents," Robbins says, noting that customer service features far more variability than other industries that have been successfully automated, such as automobile production.

"With production, to make a car it's the same process, the parts all need to come out the same, and programming the machines is always the same," Robbins says. "With this industry, humans aren't the same—they're highly volatile. The programming for [contact centers] varies; the language isn't always logical. That's where it gets more complicated."

That said, researchers are still encouraged by the progress being made in this area, and expect advances to continue.

"The amazing pace of technical innovation in the speech-to-text and text analysis area has lured researchers from diverse areas to work in fun and productive teams," says Mei Kobayashi, manager, Data Science/Text Analysis, for NTT Communications, formerly of IBM Research Japan. "There has never been a more exciting time to be working in this area."

Further Reading

White Paper: How are Artificial Intelligence & Virtual Assistance Changing the Contact Center?:
https://www.callcentrehelper.com/report.php?id=185

Let's Chat: 4 Limitations of Automated Agents in the Contact Center:
http://www.bitpipe.com/detail/RES/1483926562_479.html

Video-IPSoft's Amelia:
https://www.youtube.com/watch?v=KgSw8ckG7Jo

Keith Kirkpatrick is principal of 4K Research & Consulting, LLC, based in Lynbrook, NY.

In Memoriam | DOI:10.1145/3107180

Lawrence M. Fisher

Charles P. 'Chuck' Thacker: 1943–2017

MICROSOFT RESEARCHER Charles P. Thacker, awarded the 2009 ACM A.M. Turing Award in recognition of his pioneering design and realization of the first modern personal computer, and for his contributions to Ethernet and the tablet computer, died Monday, June 12, at the age of 74, after a brief illness.

Thacker, born in Pasadena, CA, on Feb. 26, 1943, earned his bachelor of science degree in physics from the University of California, Berkeley (UC Berkeley) in 1967.

In 1968, Thacker joined UC Berkeley's "Project Genie" to finance a graduate degree in physics. Instead, he recalled, "I went to work for this computer project," which the Berkeley Time-sharing System, commercialized by Scientific Data Systems as the SDS 940.

Thacker joined Butler Lampson (recipient of the 1992 ACM A.M. Turing Award) and others to launch the startup Berkeley Computer Corporation (BCC). While BCC was not successful, this group became the core technologists of the Computer Systems Laboratory at Xerox Palo Alto Research Center (PARC).

Thacker spent the 1970s and 1980s at PARC. There, he led the project that developed the Xerox Alto personal computer system, the first computer designed from the ground up to support an operating system based on a graphical user interface. The hardware of the Alto was designed mostly by Thacker, with Lampson developing its software.

He also is credited as co-inventor (along with Robert Metcalfe, David Boggs, and Lampson) of the Ethernet family of networking technologies, developed at PARC between 1973 and 1974.

In 1983, Thacker was part of the group of computer scientists led by Robert Taylor (manager of PARC's Computer Science Laboratory) that left PARC to found the Systems Research Center (SRC) of Digital Equipment Corp. (DEC). During his tenure there, Thacker developed Firefly, one of the first multiprocessor workstation systems.

"I have designed chips, I can design logic, I can design systems, and I can write software up to and including user interfaces."

In 1997, he joined Microsoft Research, where he helped establish Microsoft Research Cambridge at England's University of Cambridge.

Returning to the U.S., Thacker designed the hardware for Microsoft's Tablet PC, based on PARC's "interim Dynabook" (which was never built), and the Lectrice, a pen-based hand-held computer prototype developed at DEC SRC.

In 1984, Thacker, Lampson, and Taylor received the ACM Software Systems Award "for conceiving and guiding the development of the Xerox Alto System, which clearly demonstrates that a distributed personal computer system could provide a desirable and practical alternative to time-sharing." They also were named ACM Fellows in 1994 in recognition of that work.

In 2004, the National Academy of Engineering awarded Thacker, along with Alan C. Kay, Lampson, and Taylor, its Charles Stark Draper Prize "for the vision, conception, and development of the first practical networked personal computers."

In 2007, Thacker was awarded the IEEE John von Neumann Medal for his "central role in the creation of the personal computer and the development of networked computer systems."

In 2010, ACM chose Thacker to receive the 2009 ACM A.M. Turing Award "for the pioneering design and realization of the first modern personal computer—the Alto at Xerox PARC—and seminal inventions and contributions to local area networks (including the Ethernet), multiprocessor workstations, snooping cache coherence protocols, and tablet personal computers."

In an interview in the July 2010 issue of *Communications*, Thacker said, "I can lurk at a lot of different levels. I have designed chips, I can design logic, I can design systems, and I can write software up to and including user interfaces." He said his work on Ethernet at PARC, and on Firefly and fault-tolerant networks at DEC, "have a common thread, which is they are part of a distributed system—they don't stand in isolation." The Alto, he recalled, was a "nice" single-user machine, but its "real power" was unleashed by networking.

Thacker said the "secrets for his decades of continual success" included: strive for simplicity, build a kit of reusable tools, insist on sound specifications, think broadly, and make sure your collaborators also succeed.

In 2010, ACM then-president Wendy Hall said Thacker's "contributions have earned him a reputation as one of the most distinguished computer systems engineers in the history of the field. His enduring achievements—from his initial innovations on the PC to his leadership in hardware development of the multiprocessor workstation to his role in developing the tablet PC—have profoundly affected the course of modern computing."

Communications editor-in-chief Andrew A. Chien observed, "Chuck not only made seminal contributions to computer architecture, but was a tremendous inspiration to the computer systems community through the beautiful simplicity of his designs and generous mentoring of young researchers and new ideas."

—Lawrence M. Fisher

PHOTOGRAPH BY RICHARD MORGENSTEIN

DOI:10.1145/3108926

David P. Anderson

Historical Reflections
Prophets, Seers, and Pioneers

Reflections on historical prognostications for the future.

In the 1970s, Chris Evans, a psychologist and computer scientist on the staff of the National Physical Laboratory, Teddington, observed that many of the first-generation pioneers of modern computing; the people who can reasonably be crediting with laying the foundations of the digital age, were still with us. Chris conceived the idea that these people should be interviewed, and their recollections of the projects they led, the people they worked with, and the genesis of their ideas should be recorded for future generations. Tragically, aged only 48, Chris succumbed to cancer before he was able to complete the interview series he planned. Further interviews were carried out by Brian Randell, Simon Lavington, and others, but without intending any disservice to their sterling efforts, the conversations featuring Chris demonstrated standard of professionalism and ease within the milieu that really sets them apart.

The Evans interviews were released by the Science Museum, London,[a] as a set of 20 audio recordings under the title “The Pioneers of Computing.” Almost 20 years ago, I became closely involved with the “Pioneers” series, transcribing, documenting, and annotating the interviews in collaboration with the Science Museum. One of the signature features of Chris Evans’ interviewing technique was to conclude most of his recorded conversations with a couple of questions broadly concerned with prognostication. The first was to ask the interviewee to consider their state of mind at the time they were undertaking their pioneering work, and to say how they would have expected computing to have developed up to the present day (that is, the mid-1970s).

The second signature question of the Evans interviews encouraged the pioneers to look forward 10 or 20 years and indicate how they expected computing to progress into the 1990s. These questions have always struck me as being a good way to close out a somewhat technical conversation in a relaxing and informal way; more or less the complement of opening an interview by asking if the interviewee had a good journey. But quite aside from being a socially graceful exercise, the questions also produced some interesting responses. These range from F.C Williams’ typically terse and somewhat cheeky ‘no-comment’ response, “I’m not really interested in computers. I mean it’s just no good asking me a question like that. I made one, and I thought one out of one was a good score so I didn’t make any more”[b] to Allen Coombs’ much more loquacious 1,500-word response.

It is fair to say that predicting the future with any degree of accuracy is a tricky business. For my own part, my recent track record leads me to agree with Jane Austen’s Emma, who opines “I begin to doubt my having any such talent.”[c]

As part of my work I lead a research team that has a very successful track record in being awarded research grants from the European Commission, and others. This funding is critical for keeping together our in-

[a] Published by Computer Capacity Management Limited Reading and Hugo Informatics.

[b] All the quotations from the Pioneers recordings are drawn from my own transcriptions.

[c] Austen, J. *Emma*, Chapter XI.

IMAGE BY TONI V/SHUTTERSTOCK

ternational team of experts, and for recruiting new talent. The work that it supports keeps us embedded in the leading research in our field and supports international research collaboration on a reasonably large scale. For these reasons, it is important to have a keep a close eye on any broadly 'political' developments that might affect us, as well as paying attention to pure research matters. More than one year ago, as I considered how the Brexit vote would turn out, I was certain the result would be close, but I was very confident that the voters would, in the end, decide that continued membership of the EU was the wisest course of action. Areas of the country that benefitted most from regional and other support would surely realize where their interests were best served. Things would go on pretty much as they were. Having learned little from the experience of getting wrong a critically important prediction on matters where I am very well informed, I turned my attention to the 2016 U.S. Presidential election. It turns out that being relatively ignorant did nothing to improve my crystal-ball gazing.

One of the themes that emerged from the responses given to Chris Evans by the Pioneers, concerned an expectation that the costs of computing would fall. Another concerned communications infrastructure.

John M.M. Pinkerton, one of the leading figures involved with the LEO computer hit the nail fairly well on the head in saying: "Well a great many things are going to happen but the major influences I think are these: that first of all the cost of processing and the cost of storage is continuing to fall, it's fallen a lot in the last 10 years and it's continuing to fall and as far as I can see its going to go on falling. But what doesn't fall is the cost of communications, and also the cost of using people doesn't fall, it tends to go up. The result of this is there won't be anything like the obsession there has been with the efficient use of processing or storage but there will be concern over the costs of communications. Not only that, there will be a desire for everybody to have direct personal access to computing potential and because data arises everywhere and because people are everywhere we're going to have to make computing facilities available everywhere."

Arthur W. Burks, senior engineer on the ENIAC, drew a distinction between engineering and theory: "Well, we need to divide computing science into the computers proper and the theory of computers. I think for computers it is clear that they become cheaper and faster and that the revolution of computers in terms of how they interact with us and that the uses that we make of them will continue apace at least for the next 10 years. I think the theory of computers has developed much more slowly and I am not sure when there will be important breakthroughs in the theory of computers."

Konrad Zuse, the prodigious German computer pioneer contrasted the development of hardware and software, but also wanted to sound a cautionary note: "When I was a pioneer in the field and my colleagues were working in the two fields and translating one and another. Today, you have specialists for hardware and specialists for storage techniques, specialists for languages, specialists

Crossroads The ACM Magazine for Students
SPRING 2016 VOL. 22 · NO. 3
XRDS.ACM.ORG
XRDS
Digital Fabrication
A Manifest for Digital Fabrication
Soft Printing with Fabric
Fabrication Lends a Hand
WINTER 2015 VOL. 22 · NO. 2
SUMMER 20

for theoretical, informatic and so on … today we have a breakdown of the prices of the integrated circuits and the first consequence is that you can have very cheap and very small computers and this will go on. That means that processing units will become relatively cheap, I think in one or two years you will have already the machine like the Z3, I made in 1941, which to that time took a whole room, where you can put in the pocket. And this development will go on and I am not quite sure that the … hardware engineers and the software engineers, will work good enough together to take all the consequences of this development. And, I can't say exactly how the computers will look in 10 years or 20 years, but the development is not just finished not at all. We are in the full development now and the consequences of these new techniques will be that intelligence will get cheaper and cheaper. I don't think this is good. There is surely … there is some danger in this."

John V. Atanasoff remarked with great modesty: "I don't think I'm very wise. I think we see the main attributes of computers during the next years. They will become smaller, they will require less electricity. Once speaking about computers, I felt as if I should say something good to the people that were before me, and I said one thing you can say about computers is they will give great benefits without great losses of energy and I think this will one of the facets of new computers, the energy of which they use at the present time is of no consequence."

Donald W. Davies, developer of the notion of packet switching, opined: "To do this you first have to do some extrapolation of the technology, you need to know how far that's going to go and to examine, for example, how much steam there is left in semiconductor development, store development, communications development. We could start, perhaps, with communications. Here the digit rates that are available on long lines in this country can be multiplied by factors of thousands with the technology which is almost available now. In other words, there's really no limit to the digit rates available and therefore the cost per digit could, in principle, come right down. So that, high bandwidth shouldn't cost very much. This is all extrapolating and glossing over all the political and organizational problems. With regard to semiconductor technology it's not quite so clear but I think one could say that there's probably a factor of at least two and perhaps as much as 10 available in the speed, power bandwidth, and so on. Derived from things like iron implantation and so forth so that there's quite a lot to be gained but perhaps not as … not quite so much as in communications. Again, in storage—the technology is probably tied rather closely to that of semiconductors. So, in general you can say there will be cheaper machines, slightly faster machines and almost unlimited communications."

Harold L. Hazen, who contributed significantly to the theory of servomechanisms and feedback control systems, was somewhat diffident observing: "I have been so far away from active participation in these things that, and I never was a great speculator. For large extrapolations, one sees sizes going down, the limits of almost microns of element size seem not too much further compressible. Reliability is at an impressive level now, probably will go up. See, what is it? Cost per unit of computation has been halving every two or three years or something of that sort, I don't remember the exact figure. It seems that such series must become asymptotic somewhere, where that will be I'm not sure! But extrapolation for me or imagining what lies ahead would have to be based on the rather plebian and earthy process of simply extrapolating what's happened the last 10 years . When is the

It is fair to say that predicting the future with any degree of accuracy is a tricky business.

software going to be the limitation rather than the hardware? I just don't know.”

Ralph J. Slutz, who worked in the IAS and SEAC computers, looked forward to increasing miniaturization, and networking: “What I see in the very near future is a growth of more small computers, say associated with individual engineering work or laboratory work such as that which really are stand-alone computers and can work by themselves perhaps in real time with a laboratory experiment but which, when the need occurs, comes around for something bigger than they customarily can handle, can be connected to a big central computer utility. In a sense, I sort of refer to it at the present time as the invisible computer network because 99% of the time it wouldn't exist but then you dial up on a telephone line or high speed line to your computer utility and get the advantage of large facilities.”

There were a number of pioneers like Freddie Williams, who, perhaps sharing my own lack of confidence in their ability to see clearly into an uncertain future, were reluctant to say much.

Herman H. Goldstine, for example, responded: “I don't think I would even try that. Every time I've watched that kind of thing, I've seen how terribly badly the seers have missed the boat. For example, it was only a decade ago, I think, that people … everybody was saying that the terminal was going to be “the answer to a maiden's prayer.” It turns out now, I think that people want small computers which are more or less stand-alone with some capacity to be connected up to a big computer and I don't know what it will be in a decade. I just wouldn't guess.”

John W. Mauchly was similarly reluctant: “This is sort of like writing science fiction. Science fiction and the comic strips like Buck Rogers try to be way ahead of the actuality but really, we don't see much further than the end of our nose. And human beings usually extrapolate from what they now know, don't really predict anything that is so remarkable and I am afraid I have that same limitation. “

The theme of the likely emergence of learning computers and A.I., was taken up by British Pioneers who had worked with Alan Turing at Bletchley Park, and elsewhere.

I find the modestly expressed prescience of the pioneers who laid the foundations of the digital age both encouraging and uplifting.

Allen W.M. Coombs, in a very full response remarked: “Well, now I think the future of the computer lies in this. With large-scale integration we are going to be able to do this on a much bigger scale, it's getting bigger all the time … bigger and bigger in numbers and smaller and smaller in bulk, we can get closer and closer to a brain which can learn. I think probably that the next stage of the computer is to be large scale learning of really difficult and complicated things. Not just shapes, which are rather simple, but more difficult things. It is said that there are several stages of learning machine. There is the adaptive machine which can be changed … modified, that is the learning machine which really means an adaptive machine with a human teacher and there is the self-organizing machine, which is a machine which can … which is adaptive and can learn but doesn't need a human to tell it what to learn, it finds out for itself. Every child in this sense is a self-organizing system. And the next stage of computer technology is to make self-organizing systems. That is something that hasn't been done yet. A lot of people have got ideas but there is a chance of doing it now that we have got large scale integration available to us and getting more and more understanding of what goes on in a brain when it learns things.”

I.J. (Jack) Good thought that: “The main development will probably be in software I think, in machine intelligence work. But the main potential advance perhaps in electronic computers will be to go to ultra-parallel computers in my opinion, like the brain. I see that as the biggest potential advance in computers apart from the possibilities of programming reaching intelligent machinery.”

Donald Michie responded that: “The theme which I see coming to the fore is the transfer of systematized human knowledge, for example from books where most of it now is, but including in the brains of experts, into computing systems. Not just as look-up systems but in the form of operational knowledge which the computing systems can utilize to perform skilled tasks of interest to particular professional specialists. Now once that process gets under way it is inherently a tear away process, it's a bootstrapping process, because then you have machine intelligence systems which are able assistants, not only in organic Chemistry, or astronomy or whatever branch of science a particular scientist's assistant program has been engineered for, but don't forget that there is one other branch of science namely machine intelligence and that branch will also acquire powerful and teachable and self-adaptable research assistants and that is the beginning of a tear away process. And I would say myself that sometime between 1980 and 1985, I would expect it to spread through the science community, the realization for good or otherwise that this tear away process has now started.”

Perhaps the most remarkable thing about these various attempts to look into the future, is the very great extent to which all of the people who ventured an opinion got nearly everything right. We live in a time when provable falsehoods are presented as ‘alternative facts’ and expertise is routinely disparaged. In that context, I find the modestly expressed prescience of the pioneers who laid the foundations of the digital age both encouraging and uplifting. These voices from the past, captured by Chris Evans, give me renewed hope for the future.

David P. Anderson (cdpa@btinternet.com) is Professor of Digital Humanities at the Centre for Research & Development (Arts)/Cultural Informatics Research Group, University of Brighton, U.K.

DOI:10.1145/3108928

Jennifer Wang

Education

Is the U.S. Education System Ready for CS for All?

Insights from a recent Google-Gallup national research study seeking to better understand the context of K–12 CS education.

The increasing focus on K–12 computer science (CS) education in the last few years has been driven by two key factors: producing enough computing professionals to support the workforce and drive innovation, and ensuring that this workforce is sufficiently diverse to represent all perspectives. However, the diversity gap persists. Stereotypes and educator biases start young, as early as preschool.[1–3] One way Google is contributing to CS education efforts is through new research that identifies structural and social barriers, as well as strategies to overcome them.

This column presents insights derived from the Google-Gallup national research. With CS education relatively young in the K–12 space, our multiyear study sought to understand the context of K–12 CS education. Over two years, we surveyed about 16,000 nationally representative students, parents, teachers, principals, and superintendents across the U.S. in the fall/winter of both 2014–2015 and 2015–2016. Students, parents, and teachers were surveyed via telephone while principals and superintendents were surveyed online via an email invitation. We also designed the study with a focus on diversity—particularly girls, Blacks/African Americans, and Hispanics/Latinx. We wanted to uncover any structural barriers as well as social perceptions and biases that may be affecting these groups.

Members of an Iowa elementary school coding club demonstrate projects during a May 2017 campaign promoting more computer science education.

PHOTO BY PHIL ROEDER/FLICKR

We found that overall, there was large demand and interest in CS[a] education among students, parents, and educators. Specifically, 82% of students were at least somewhat interested in learning CS, with Black and Hispanic students 1.5 and 1.7 times as likely as white students, respectively, to be very interested in learning CS.[b] Among parents, 91% wanted their children to learn CS, with 84% indicating CS is at least as important as required subjects like math, science, history, and English and 24% indicating CS is *more* important than these required subjects. Approximately 60% of educators agreed that CS should be required if available. In fact,

[a] The survey provided a definition of CS and reminded respondents multiple times throughout: "Computer science involves using programming/coding to create more advanced artifacts, such as software, apps, games, websites and electronics, and computer science is not equivalent to general computer use."

[b] Students indicated whether they were "very," "somewhat," or "not at all" interested. The full research report from which this column was developed is available at http://atg.co/cseduresearch.

lower-income parents and teachers at schools with greater proportions of students receiving free-/reduced-lunch were more likely to value CS learning.

Despite this high demand from parents, our study revealed structural barriers at school and at home. Even though our study found an increase in the percentage of K–12 principals reporting their schools offer CS with programming/coding—from 25% in 2014–2015 to 40% in 2015–2016—we saw persistent disparities in who had access to and who had learned CS. Black students reported significantly less access to classes with CS compared to white students (47% vs. 58%). Also, lower-income students were less likely to report access to CS learning opportunities in general. Hispanic and Black students reported less exposure to computers at school and at home compared to white students (30% of Black and 26% of Hispanic vs. 45% of white students reported using computers every day at home; 34% of Black and 31% of Hispanic vs. 42% of white students reported using computers every day at school), and Hispanic students were less likely to indicate they knew someone who worked in tech (49% of Hispanic students vs. 68% of white and 65% of Black students). Girls were less likely to say they had learned CS (50% of girls vs. 59% of boys), while boys were more likely to say they learned CS on their own (of students who have learned CS, 54% of boys vs. 41% of girls).

Interestingly, we also found that regardless of race/ethnicity or gender, 80% of students who have learned CS said that they learned CS in a class at school, about twice the rate of any other means of learning, including on their own, through afterschool clubs, online, or in any other program outside of school. This data strongly suggests formal education remains the best way to ensure widespread and equitable access to CS learning.

Yet, we found schools faced many barriers to offering CS classes. We asked principals and superintendents why they did not offer CS in their schools and districts. The most commonly cited barriers had to do with lack of qualified teachers and competing demands of standardized test preparation. Lack of qualified teachers was cited by 63% of principals and 74% of superintendents. Not enough funding to train teachers was cited by 55% of principals and 57% of superintendents. The need to devote time to testing requirements was cited by 50% of principals and 55% of superintendents. This indicates computing professionals can play an important role in expanding access to CS by supporting organizations that train teachers and by providing mentoring and resources to teachers and students.

> **These findings point to proactive ways that computing professionals (both industry and academic) can support and advocate for CS education.**

Despite the high demand and interest among parents, nearly half of administrators (principals and superintendents) cited a lack of demand from parents as a barrier to offering CS. In fact, less than 8% of administrators believed parent demand was high and less than 30% of educators agreed that CS was a top priority at their school or district. We found that even though 91% of parents wanted their child to learn CS, less than 29% of parents reported they have expressed support for CS to their schools. This lack of direct parent engagement indicates that parents feel unprepared or unable to effectively advocate with school administrators on behalf of their children.

Finally, we also identified social barriers in CS education. Not surprisingly, students and parents see media images of those who practice CS as mostly male, white, and wearing glasses. And these perceptions expanded beyond media to beliefs among students, parents, and educators. The belief that you need to be "smart" prevailed—49% of students and 57% of parents reported that "people who do CS need to be very smart" while 62% of teachers and 56% of principals agreed that students good at math and science are more likely to succeed in

Calendar of Events

August 7–11
SIGIR '17: The 40th International ACM SIGIR Conference on Research and Development in Information Retrieval
Shinjuku, Tokyo, Japan,
Sponsored: ACM SIG,
Contact: Hiroaki Ohshima,
Email: ohshima@ai.u-hyogo.ac.jp

August 11–13
SIGDOC '17: The 35th ACM International Conference on the Design of Communication
Halifax, NS, Canada,
Sponsored: ACM SIG,
Contact: Rebekka Andersen,
Email: randersen@ucdavis.edu

August 13–17
KDD '17: The 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining
Halifax, NS, Canada,
Co-Sponsored: ACM SIG,
Contact: Stanislaw J. Matwin,
Email: stan@site.uottawa.ca

August 17–21
ICER '17: International Computing Education Research Conference
Tacoma, WA,
Sponsored: ACM SIG,
Contact: Donald Chinn,
Email: dchinn@u.washington.edu

August 21–25
SIGCOMM '17: ACM SIGCOMM 2017 Conference
Los Angeles, CA,
Sponsored: ACM SIG,
Contact: K.K. Ramakrishnan,
Email: kk@cs.ucr.edu

learning CS. Gendered stereotypes also persisted, with 74% of students, 64% of parents, and 63% of teachers indicating that boys are more interested in CS than girls and 44% of students, 37% of parents, and 36% of teachers saying that boys are more likely to succeed in learning CS compared to girls.

The stereotypes played out in who is and is not encouraged—only 26% and 27% of girls reported being told by a teacher or parent, respectively, that they would be good at CS compared to 39% and 46% of boys. Girls were also less likely to see role models in the media, with 31% of those who see CS in the media saying they never see someone "like me" engaged in CS, nearly twice the rate of boys. In a previous study,[4] we found that encouragement was the biggest factor influencing whether girls intended to pursue CS or not. In fact, our Google-Gallup study found that girls not only had lower confidence in learning CS (48% of girls vs. 65% of boys were very confident they could learn CS), they were also less likely to be aware of CS learning opportunities outside of classes at school and were less likely to learn CS at all (50% of girls vs. 59% of boys). Anyone can help challenge these stereotypes by promoting broader perceptions of those who engage in CS and building more inclusive environments in computing.

While we see large support for CS education among students, parents, and educators, many barriers stand in the way. What can ACM members do? Education is a deeply rooted system, so solutions need to tackle these challenges from both the bottom up and the top down and through structural and social modes.

Our Google-Gallup research uncovered a number of persistent barriers that hamper the education system's readiness of CS for All students, and these findings point to proactive ways that computing professionals (both industry and academic) can support and advocate for CS education. To tackle structural barriers and increase access, companies and computing professionals can speak up and advocate for CS in their communities and local schools to amplify students' and parents' interest and value of CS learning. From the top down level, they can call state legislators and ask for support for CS in their state — for example, share state data (see https://goo.gl/EjiVFp), Google's policy advocacy brief (see https://goo.gl/gzVOlf) and resources on code.org/promote (see https://code.org/promote).

However, increasing access alone will not enable all students to learn CS; we need to also remove social barriers. To ameliorate social challenges and enable diverse students to take advantage of available opportunities, computing professionals can spend time with teachers and students, particularly those who would otherwise not engage in CS. They can also avoid unintentionally supporting stereotypes with comments about who and what is nerdy or geeky. Rather, it is important to emphasize computing as advancing a variety of industries, from healthcare to agriculture to the arts, as well as the diversity of backgrounds of the people who practice computing. As influential adults, we can each do the small gesture of encouraging students no matter where they come from or who they are, and supporting them through positive, inclusive environments.

This is only the beginning of the journey. Increasing access to CS learning is an early needed step, but without individual and societal support to remove social barriers, the diversity gap will persist in our education system. Computing professionals can play an essential and active role in making CS accessible for all students. They can advocate with school leaders and help empower and enable young people. Each of us has the power to help build and strengthen the computing field by encouraging and welcoming all K–12 students. C

References

1. Bian, L., Leslie, S.J., and Cimpian, A. Gender stereotypes about intellectual ability emerge early and influence children's interests. *Science 355* 6323, (2017), 389–391.
2. Cvencek, D., Meltzoff, A.N., and Greenwald, A.G. Math–gender stereotypes in elementary school children. *Child Development 82*, 3 (2011), 766–779.
3. Gilliam, W.S. et al. Do Early Educators' Implicit Biases Regarding Sex and Race Relate to Behavior Expectations and Recommendations of Preschool Expulsions and Suspensions? Yale Child Study Center (Sept. 2016), 991–1013.
4. Google Inc. Women Who Choose Computer Science—What Really Matters. 2014; https://goo.gl/rLX6ax.

Jennifer Wang (jennifertwang@google.com) is the Research Program Manager on Google's Computer Science Education Team.

DOI:10.1145/3108930

George V. Neville-Neil

Kode Vicious

The Observer Effect

Finding the balance between zero and maximum.

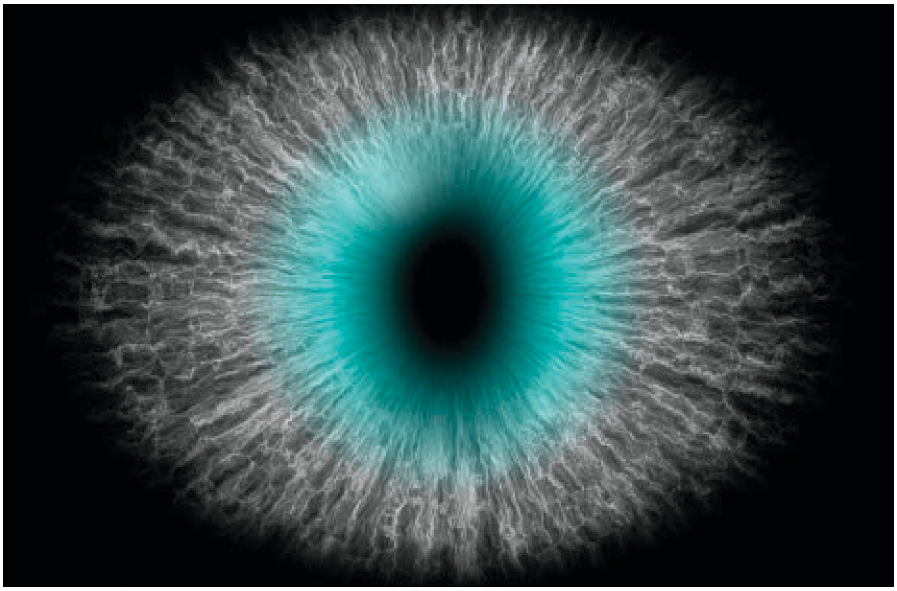


IMAGE FROM SHUTTERSTOCK.COM

Dear KV,

The company I work for rolled out a new monitoring system one weekend, and it didn't go as well as we would have liked. When we first brought up the monitoring system, several of our servers started to show very high CPU load. Initially, we could not figure out why. The monitoring processes on each server were very busy, so we turned off the monitoring system and the servers got less busy. Eventually, we realized it was the number of polls being issued by the monitoring system that was causing the servers to use so much CPU time. We decreased the polling frequency to every 10 minutes, and this seemed to be the sweet spot for system performance. What I would like to know is how one should go about tuning such systems, as it seems still to be done via trial and error.

Polled Too Frequently

Dear Polled,

Trial and error? The problem here is usually a failure to appreciate just what you are asking a system to do when polling it for information. Modern systems contain thousands—sometimes tens of thousands—of values that can be measured and recorded. Blindly retrieving whatever it is that might be exposed by the system is bad enough, but asking for it with a high-frequency poll is much worse for several reasons.

The first reason is the one that you bring up in your letter: the amount of overhead introduced by simply asking for the data. Whenever you ask the system for its configuration state, whether that's a routing table or the state of various sysctls (system control variables), the system has to pause other work to provide a consistent picture of what's going on. KV knows that in recent years the idea of consistency has been downplayed in favor of performance—in particular, by various database projects. In the systems world, however, we still think that consistency is *a good thing*™ and therefore the system will try either to snapshot the data you request or to pause other work while the data is read out. If you ask for a few thousand items, and a random `sysctl -a` shows 9,000+ elements on a server I am using, then that is going to take time—not forever but not nothing, either.

The second reason that polling for data frequently is a problem is that it actually hides the information you might be looking for in the noise generated by retrieving and communicating the values you asked for. Every time you ask the system for some stats, it has to do work to get those stats, and the system doesn't account for your request separately from any other work it has to do. If your monitoring system is banging away at the server asking for data every minute, then what you will see in your monitoring system is the load that the system itself is generating. Such Heisen-monitoring, where your monitoring system is overwhelmingly affecting the measurements, is completely pointless.

In a monitoring system, there is always the tension between too much and too little information. When you're debugging a problem, you always wish you had more data, but when your system is running normally, you want it to do the work for which it was deployed. Unless you enjoy just pushing monitoring systems—and, yes, there is definitely a handle for those people somewhere on social media—you need to find the Goldilocks zone for your monitoring system. To find that zone, you must first know what you're asking for. Figure out which commands the monitoring system is going to execute on your servers, and then run them individually in a test environment and measure the resources they require. You care about runtime, which can be found to a coarse level with the time(1) command. Here is an example from the server just mentioned.

```
time sysctl -a > /dev/null
sysctl -a > /dev/null 0.02s
user 0.24s system 98% cpu
0.256 total
```

Here, grabbing all of the system's various system-control variables takes about a quarter of a second of CPU time, most of which is system overhead—that is, time spent in the operating system getting the information you requested. The time(1) command can be used on any utility or program you choose.

Now that you have a rough guess as to the amount of CPU time that the request might take, you need to know how much data you're talking about. Using a program that counts characters, such as wc(1), will give you an idea of how much data you're going to be gathering and moving off the system for each polling request.

```
sysctl -a | wc -c
378844
```

You would be grabbing more than a quarter of a megabyte of data here, which in today's world isn't much, but it still averages out to 6,314 bytes per second if you poll every minute; and, in reality, the instantaneous rate is much higher, causing a 3Mbps blip on the network every time you request those values.

Of course, no one in his or her right mind would just blindly dump all the sysctl values from the kernel every minute—you would be much more nuanced in asking for data. KV has seen a lot of unsubtle things in his time, including monitoring systems that were set up to do just this sort of ridiculous level of monitoring. "We don't want to lose any events; we need a transparent system to find bugs!" I hear the DevOps folks cry. And cry they will, because sorting through all that data to find the needle in the noise will definitely not make them happier or give them the ability to find the bug.

What is needed in any monitoring system is the ability to increase or reduce the level of polling and data collection as system needs dictate. If you are actively debugging a system, then you probably want to turn the volume of data up to 11, but if the system is running well, you can dial the volume back down to 4 or 5. The volume can be thought of as the polling frequency times the amount of data being captured. Perhaps you want more frequent polling but less data per request, or perhaps you want more data for a broader picture but polled less frequently. These are the horizontal and vertical adjustments you should be able to make to your system at runtime. A one-size-fits-all monitoring system fits no one well. The fear, of course, is that by not having the volume at 11 you will miss something important—and that is a valid fear—but unless the whole reason for your existence is to capture all events at all times, you will have to find the right balance between 0 and maximum volume.

KV

Related articles on queue.acm.org

Scaling in Games and Virtual Worlds
January 02, 2009
http://queue.acm.org/detail.cfm?id=1483105

Kode Vicious Bugs Out
Tackling the uncertainties of heisenbugs
http://queue.acm.org/detail.cfm?id=1127862

A Conversation with Bruce Lindsay
Designing for failure may be the key to success.
http://queue.acm.org/detail.cfm?id=1036486

Software Needs Seatbelts and Airbags
Emery D. Berger

Finding and fixing bugs in deployed software is difficult and time-consuming. Here are some alternatives.
http://queue.acm.org/detail.cfm?id=2333133

George V. Neville-Neil (kv@acm.org) is the proprietor of Neville-Neil Consulting and co-chair of the ACM *Queue* editorial board. He works on networking and operating systems code for fun and profit, teaches courses on various programming-related subjects, and encourages your comments, quips, and code snips pertaining to his *Communications* column.

DOI:10.1145/3107924

Dominic Horsman, Vivien Kendon, and Susan Stepney

Viewpoint
The Natural Science of Computing

As unconventional computing comes of age, we believe a revolution is needed in our view of computer science.

TECHNOLOGY CHANGES SCIENCE. In 2016, the scientific community thrilled to news that the LIGO collaboration had detected gravitational waves for the first time. LIGO is the latest in a long line of revolutionary technologies in astronomy, from the ability to 'see' the universe from radio waves to gamma rays, or from detecting cosmic rays and neutrinos (the Laser Interferometer Gravitational-Wave Observatory—LIGO—is an NSF-supported collaborative effort by the U.S National Science Foundation and is operated by Caltech and MIT). Each time a new technology is deployed, it can open up a new window on the cosmos, and major new theoretical developments can follow rapidly. These, in turn, can inform future technologies. This interplay of technological and fundamental theoretical advance is replicated across all the natural sciences—which include, we argue, computer science. Some early computing models were developed as abstract models of existing physical computing systems. Most famously, for the Turing Machine these were human 'computers' performing calculations. Now, as novel computing devices—from quantum computers to DNA processors, and even vast networks of human 'social machines'—reach a critical stage of development, they reveal how computing technologies can drive the expansion of theoretical tools and models of computing. With all due respect to Dijkstra, we argue that computer science is as much about computers as astronomy is about telescopes.

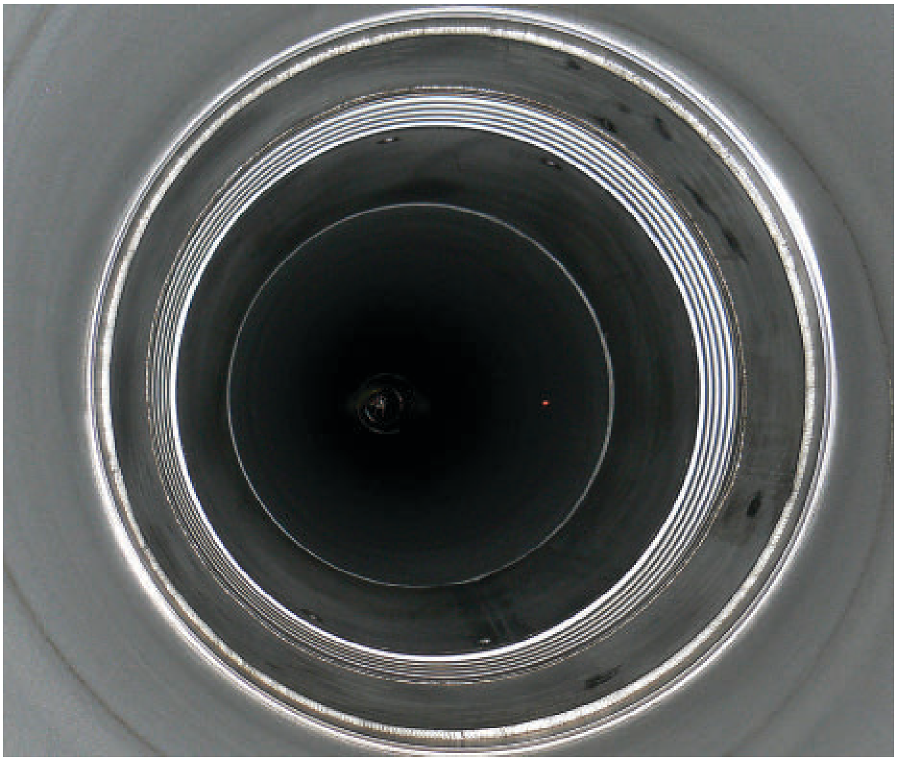


Image of the Laser Interferometer Gravitational-Wave Observatory 40m beam tube.

PHOTO BY TOBIN FRICKE/FLICKR

Non-standard and unconventional computing technologies have come to prominence as Moore's Law, that previously relentless increase in computing power, runs out. While techniques such as multicore and parallel processing allow for some gains without further increase of transistor density, there is a growing consensus that the next big step will come from technologies outside the framework of silicon hardware and binary logic. Quantum computing is now being developed on an international scale, with active research and use from Google and NASA as well as numerous universities and national laboratories, and a proposed €1 billion quantum technologies flagship from the European Commission. Biological computing is also being developed, from data encoding and processing in DNA molecules, to neurosilicon hybrid devices and bio-inspired neural networks, to harnessing the behavior of slime molds. The huge advance of the internet has enabled 'social machines'—Galaxy Zoo, protein FoldIt, Wikipedia, innumerable citizen science tools—all working by networking humans and computers, to perform computations not accessible on current silicon-based technology alone.

What all these devices, from the speculative to the everyday, share is that they currently lie beyond the reach of conventional computer science. Standard silicon-based technology is built on a toolkit of theoretical models and techniques, from lambda calculi through to programming, compilation, and verification. These tools seem to be largely inaccessible to the new technologies. How do you program a slime mold? What is the assembly language of protein folding? How do you compile for a human in a social machine? New technologies may be one or more of stochastic, continuous time, continuous space, sloppy, asynchronous, temperature dependent, sub-symbolic, evolving systems, with computationally complex encodings and decodings, and one shot construction-and-execution.

Without the ability to define and characterize how and when computing is happening in these systems, and then to import or develop the full suite of theoretical tools of computer science, we claim that the information-processing capabilities of these devices will remain underexploited. We believe we need an extended computer science that will enable us to treat these systems with theoretical and practical rigor to unlock their potential, and also to enable us to combine them with existing technology to make scalable and hybrid devices.

It is inefficient (or simply impossible) to impose the standard computing framework on many nonstandard systems.

Computer science has historically been conceived and developed around abstract Turing Machines and equivalent calculi. This discrete, symbolic logical, deterministic underlying model is realized equivalently, but differently, in one specific technology, the von Neumann stored program architecture. This technology has proved so successful, and is now so ubiquitous, that other models of computing have tended to be ignored; one example is Shannon's largely forgotten GPAC computational model, based on the technology of differential analysers. As a consequence of using only a single model, standard approaches to computing abstract away the physical implementation, leaving a theoretical computer science that is frequently viewed *as* a branch of mathematics, rather than as a physical science that is expressed *in* mathematical language. With little connection to actual physical devices, this theoretical framework can be at a loss when faced with nonstandard computing systems. Often the response is to impose top-down a standard bit-and-logic-gate framework, in the belief that this is *the* way to compute. The delicate systems in a quantum computer, for instance, can be forced to act like standard bits obeying classical logic. However, these devices gain their real power when allowed to act as natively 'quantum bits,' or qubits, with their own quantum logic gates. It is as inefficient (or simply impossible) to impose the standard computing framework on many nonstandard systems as it would be to use a sophisticated optical telescope to detect cosmic neutrinos. We do not believe that we can unlock the true potential of unconventional systems by forcing them into the mold of standard computing models.

While traditionally computer science tends to view itself as a branch of mathematics, the field of unconventional computing has tended to go too far the other way, seeing computing merely as an outgrowth of physics, or chemistry, or biology. Arguments around computing power often overfocus on the physical *theory* of the device, rather than what can be implemented in the physical device itself. Neural nets, for example, can be modeled using real-valued activation functions, and arguments have been made that these networks are in actuality computing those real values to arbitrary precision, and hence far outperforming the capabilities of standard computers. In practice, however, such purely abstract infinite real-valued precision is completely outside the physical capabilities of the device: it can neither be observed, nor exploited.

Computing theory should not be imposed top-down without taking into account the physical theory of the device: computer science is not mathematics. The computing ability of the system is not always identical with the computing capability of the physical device theory: computer science is not physics. What is it, then? We believe that it has features of both, consisting in the complex interplay of mathematics and physical theory through a crucial relation: representation.

Understanding computers can be seen as the key to understanding computer science. A computer crosses the boundary between the abstract domain of logic/computation, and the physical realm of semiconductors or quantum ions or biological molecules; and it does so in a way that we can precisely characterize. Consider part (a) of the the figure here, in which a 'compute cycle' starts with an abstract problem, such as adding two numbers, or finding prime factors, or calculating a shortest path. Usually expressed in some high-level language, this is then *encoded* into the computer's native language. This encoding is still in essence an abstract process: the description of the computation has been transformed from one language to another. Now the actual computer is brought in, and the native language input is *instantiated* in the target physical device. The device is configured, and the physical processes of computing initialized. Then the computer runs, as a *physical* process with a physical output in the final state of the computer. To find the output of the computation, we ask to what abstract state the physical one corresponds: which state of the program is *represented* by the physical state of the computer? This is then (abstractly) *decoded* from the abstracted output to a language to answer the original problem. The computer has output the solution.

If a computer is a good one, and running without errors, the aim of the compute cycle is to parallel the physical and abstract behaviours. The solution is an abstract answer to an abstract question; were it possible to "run" the program entirely abstractly, then the solution could be found without including any physical device in the cycle, be that an engineered computer, or a pencil-and-paper based human emulation. Computers are used as a physical proxy for this abstract mapping. A good computer is engineered such that the result of letting the physical dynamics run will parallel the abstract behavior of the program. A computer is a device that manipulates the physical instantiation of abstract concepts, in order to solve problems. It is not identical with a computation: computation is abstract, a computer is physical, and they relate through (nontrivial) representation and instantiation.

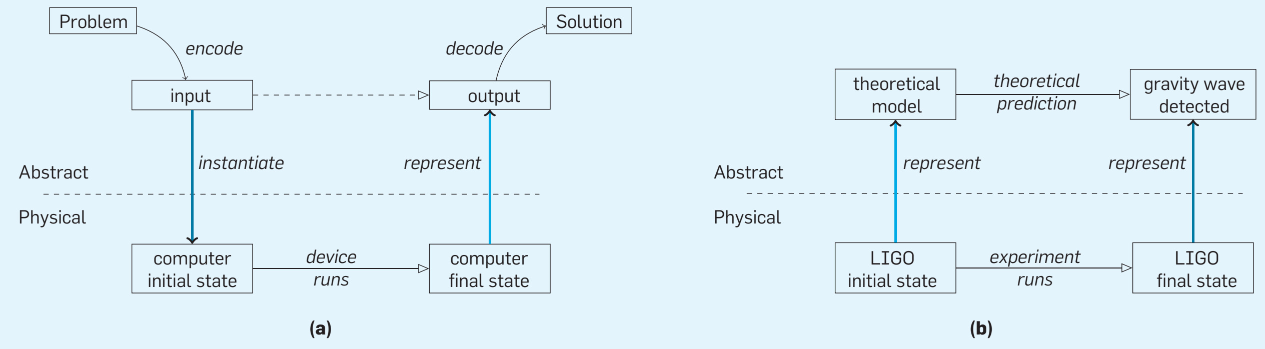
This centrality of representation is the core of a new formalism developed by the authors: Abstraction/Representation Theory (AR theory). With diagrams such as part (a) of the figure here, and an associated algebraic-like structure, AR theory is a toolkit for the foundations of computer science, and beyond. The complex interplay of mathematics, physical theory, and representation is not confined to the field of computing. It also drives the mechanism of experimental testing of abstract theories throughout the natural sciences. We can, for instance, give a diagram for the relation between astronomy and telescopes, with crucial similarities and differences to computing. Part (b) of the figure shows how theory and experiment relate in the LIGO experiment. Again abstract and physical are parallel, but now the process of running the experiment starts with the physical apparatus, rather than with an encoding of an abstract computational problem. There is an abstract representation of the apparatus in the theory of gravitational waves: that it can detect them. Also in the abstract realm there is a theoretical prediction for how the apparatus will behave if such waves indeed exist. If the experiment is successful, as with LIGO, then the theory and the abstract interpretation of the physical outcome coincide up to some error margin ε. If a theory is sufficiently good, the physical system can be removed altogether: abstract theory can be used to predict physical behavior.

Looking at these two diagrams, we uncover a deep truth. Just as a mathematical theory allows us to predict physical behavior, in a computer the physical behavior of a device is used to 'predict' the result of an abstract computation. Computing and natural science are fundamentally linked; the link is technology. Notice the direction of the arrows of representation in the two diagrams. In an experiment, they go only one way, upward: this is the *representation* of physical systems by an abstract model. In computing there is another type of relation: *instantiation* of abstract theory in physical systems. Instantiation is more complex than modelling, and requires *engineering* to construct a system that, when modeled, gives the abstract specification to be instantiated; this engineering in turn requires a sufficiently good scientific understanding of the system's properties. Not all abstract systems that can be imagined denote something in the physical world ("the present king of France"), or can be physically instantiated (faster-than-light travel).

Just like a telescope, a computer is a highly engineered device. LIGO went through many years of testing of its various components before scientists were happy that it would function as a gravitational wave detector. With the tests complete, it can now be used as a telescope to observe the universe in terms of those ripples. Similarly, computers require engineering before they can be used for computation: we need to be confident that their physical behavior parallels that of the abstract program so that the device can be used to predict its outcome (there can be engineering bugs in hardware). Computers start with computer *science*: with experiments on novel substrates and with new ways of performing computing. Only once that cycle is complete, and we know enough about how the system behaves, can a new device be engineered to instantiate a computation.

What does AR theory mean for our understanding of computer science? We claim that we now have a way to understand that computational logic arises from the physical structure of a potential computing substrate, and that it may vary widely across different classes of substrate. Computer science, in addition to its theoretical component, covers both the experimentation

Computers and telescopes: The interplay of abstract theory/programming and physical devices in (a) computing and (b) physical sciences (given here by the LIGO experiment).



and engineering phases of computing, as well as the eventual use in deployment as a computer. This understanding tells us to use an experimental and engineering process when developing new formal models and methods of computer sciences for our new devices, paralleling the process of developing new models and instruments to tackle new phenomena in rest of the natural sciences. A computational logic for a system arises, but we then abstract away from the specific device to a formal model of it. Programming these new devices is then a matter of looking for a natural internal process logic of the system, as opposed to forcing a one-size-fits-all model of computation onto some candidate computing system. Rather than looking to impose top-down the machinery of standard logic gates, we should look at the natural behaviour of the system and what 'gates' or subroutines or problem-solving it is intrinsically good at. By extracting an intrinsic computational logic of their physical components we can harness the true potential of unconventional computers.

Using our physical understanding of a substrate to inform a computational logic does not mean that such a logic is the only one possible. Just as a quantum computer can run as either quantum or classical, other non-standard systems may be capable of supporting multiple computational models. This again is found throughout the natural sciences: for example, in physics a particular system might be modelled as a continuous fluid, or as a collection of discrete particles. With different potential computational representations of a system under investigation, the key is to extract out the ones that do something useful and novel and better than other substrates—and then use that computational theory to engineer our next generation of computers.

We can then go further. With an abstract computational language that describes the native operation of unconventional devices, we would then have a logical language in which to describe the physical systems themselves, even outside a specifically computational device. Computer science could then provide high-level logical process languages for physics, chemistry, and biology—and even, with the interactions of social machines, for networks of human beings. We believe this could be of immediate practical importance to scientists in those areas, enabling them to describe high-level functioning of complex systems, and to find new and unforeseen connections between disparate systems and scenarios. These process languages could be as revolutionary for the physical sciences as for computer science.

Computers have come a long way since the days of valves and punched cards. Now computer science itself is branching off in new directions with the development of unconventional computing technologies. As the domain of computer science grows, as one computational model no longer fits all, its true nature is being revealed. Just like astronomy, computer science could describe physical systems in abstract language with predictive power, and thereby drive forward the dual interplay of technology and theoretical advancement. New computers could inform new computational theories, and those theories could then help us understand the physical world around us. Such a computer science would indeed be a natural science. C

Further Reading

Copeland, J. et al. Time to reinspect the foundations? *Commun. ACM 59*, 11 (Nov. 2016), 34–36.

Horsman, C. et al. When does a physical system compute? In *Proceedings of the Royal Society of London*, 470:20140182, 2014.

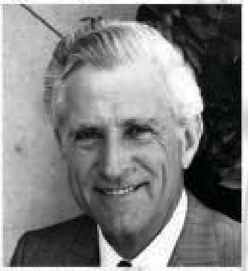
Horsman, D.C. Abstraction/Representation Theory for heterotic physical computing. In *Philosophical Transactions of the Royal Society*, 373:20140224, 2015.

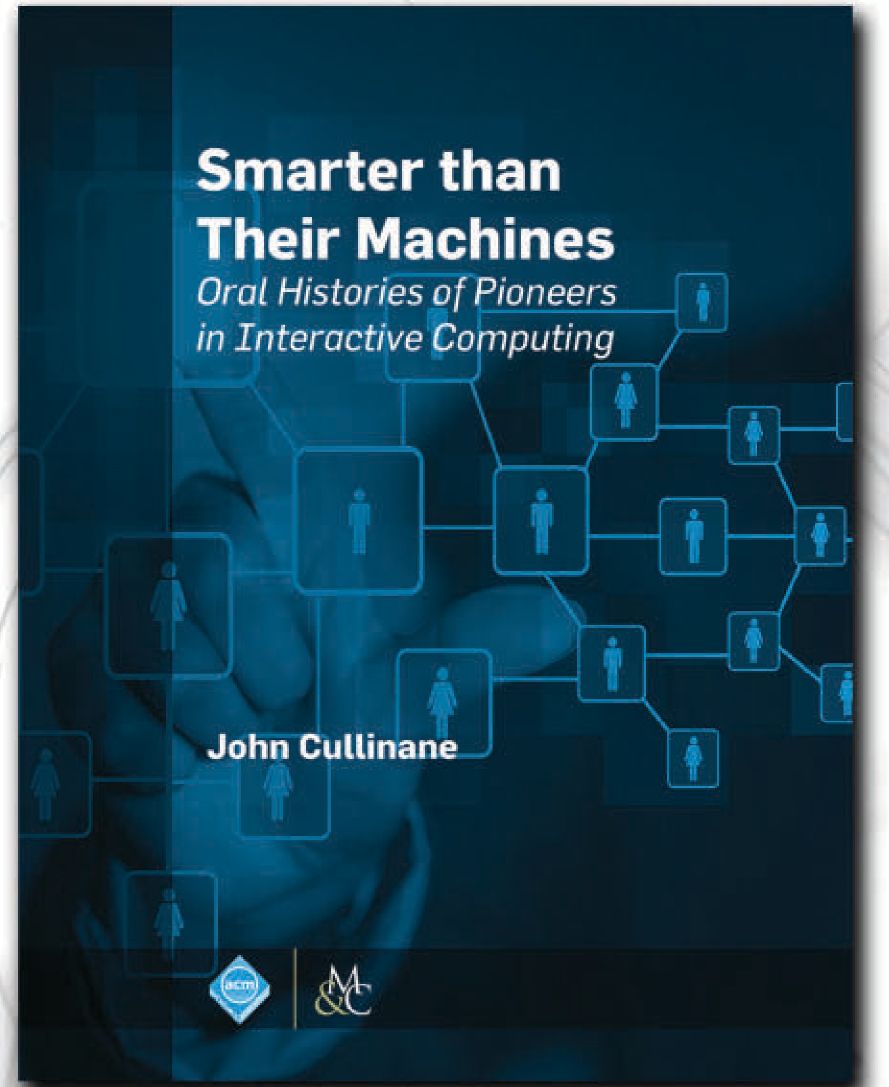
Horsman, D.C. Abstraction and representation in living organisms: When does a biological system compute? In G. Dodig-Crnkovic and R. Giovagnoli, Eds. *Representation and Reality in Humans, Animals, and Machines*. Springer, 2017.

Dominic Horsman (dominic.horsman@durham.ac.uk) is a Postdoctoral Research Associate at the University of Durham, U.K.

Vivien Kendon (viv.kendon@durham.ac.uk) is a Reader in the Department of Physics at the University of Durham, U.K.

Susan Stepney (susan.stepney@york.ac.uk) is Professor of Computer Science in the Department of Computer Science, University of York, U.K.

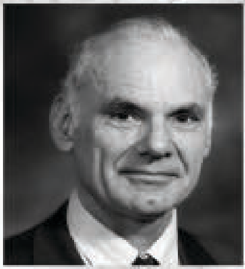




Smarter than Their Machines

Oral Histories of Pioneers in Interactive Computing

John Cullinane



DOI:10.1145/2983527

Word processors now make it possible for many authors to work on the same document concurrently. But what can they actually do?

BY RICARDO OLENEWA, GARY M. OLSON, JUDITH S. OLSON, AND DANIEL M. RUSSELL

Now That We Can Write *Simultaneously*, How Do We Use That to Our Advantage?

MODERN WORD PROCESSORS, like Microsoft Word OneDrive, SharePoint, and Google Docs, allow people to work on the same document at the same time. While systems that allow simultaneous writing have been demonstrated in research labs for some time, only relatively recently have such systems been available commercially for widespread use. Google Docs, for example, enables people to be at the same place in the document while adding, deleting, and moving text at will. Google Docs is very popular (for example, there are over 60 million users of Google Apps for Education worldwide and two million businesses use Google Apps for Work) and many people see the simultaneous writing feature as a great asset.

For example, when we analyzed in detail the writing patterns of undergraduate students enrolled in an advanced university course on project management, 95% of the documents exhibited simultaneous writing. Three documents were *only* written simultaneously.[9]

We, the four co-authors of this article, have experienced numerous sessions where simultaneous writing created notable benefits. Given that we can work simultaneously now, how can we harness that capability to make the work more efficient? What can you do with simultaneous writing? When might you *not* want to write simultaneously?

To answer these questions, we collected our stories, grouped them, and noted patterns across them. Each of the stories is told with the voice of one of the authors except the last, which is co-written by all four of us because it reflects how we wrote *this* article. Subsequently, we note the similarities and differences among our stories, and with "plural unity" created the scheme of six patterns and two epiphenomena. We believe the stories will serve as an inspiration to readers to work in new, beneficial ways.

Google Docs is relatively new and although some of our stories refer to the use of Docs, a number of the stories involve research prototypes or early commercial systems, some going as far back as the 1980s. Before we get to the stories, we first provide brief descriptions of the characteristics of each of these systems, listed in order of creation.

The Systems We Refer To

IDE, for Instructional Design Environment,[10] was a tool to create instructional content by helping designers organize and create materials rapidly. IDE was built on top of the Notecards hypermedia system.[5,6] IDE featured "cards" with different kinds of links between sections of text. While IDE was not originally intended as a multi-person collaborative system, people quickly realized they could partition work among a number of writers and get things done more quickly by working on different cards simultaneously. When someone

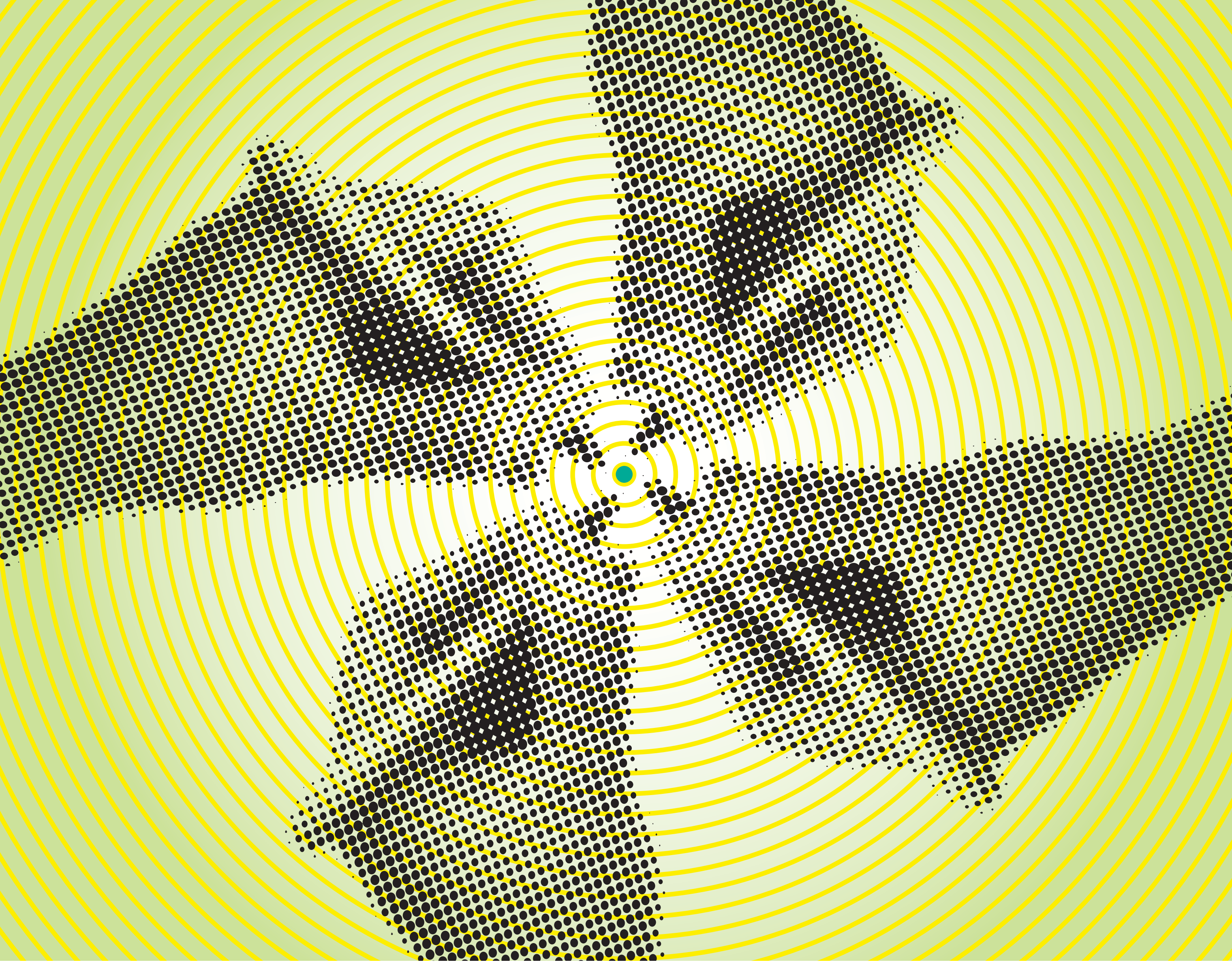


IMAGE COMPOSITION BY ANDRIJ BORYS ASSOCIATES, USING SHUTTERSTOCK

was working on a card, it was locked to the others. While this is technically not the kind of simultaneous editing that other tools have, the story where this was used for parallel simultaneous editing has some valuable lessons.

ShrEdit was a collaborative writing tool built in 1990.[7,8] A shared document was hosted on a server and individuals accessed it from client machines on the same local network. The architecture allowed multiple people to write in the same document at the same time. ShrEdit had "selection locking," meaning that one could write simultaneously within one character of one another.

Aspects was a commercial product available in the 1990s.[4] Like ShrEdit, people could edit at the same time within one character of each other. Whereas ShrEdit supported only text, Aspects supported spreadsheets, drawings, and presentations in addition to text, similar to Google Apps but running on a local network and only available for Macintoshes.

Centra Symposium was a commercial system that allowed for audio and video conferencing as well as sharing of an object, like a document or a presentation. One could allow others to edit the shared object.

Google Docs is a commercial, multiuser, shared document editing system provided by Google beginning in 2006. Docs lets multiple people edit a document simultaneously, view the revision history of the document, and share the doc with specific people or broadly on the open internet.

The Power of Simultaneous Writing

We now move to telling our stories about how the power of simultaneous writing creates significantly more productive work settings. The stories that follow are grouped into five categories:

- Writing large documents (Stories 1–3)
- Writing short documents for class assignments (Story 4)
- Displaying and collaboratively creating meeting minutes (Stories 5–6)
- Teaching through shared documents (Story 7)
- Writing this document. (Story 8)

Story 1: Doc Build-It using Google Docs, told by Ricardo. To be useful to others, software needs to be documented. Many times software needs to be rewritten from scratch because the original could not be found or once found, could not be understood. Unfortunately, many software developers hate to document. Although technical writers, like me, can do the job, there are too few of us, and the process of understanding a technology and writing appropriate documentation can take years. Consequently, only core technologies get documented.

To solve this problem, I developed a documentation method called the Doc Build-It. A Doc Build-It is a single-day event where a small group of engineers gather to simultaneously write documentation about a specific piece of technology. The Doc Build-It is a constrained activity, carefully designed to enable engineers to impart their expertise in a way that seems natural to them.

The Doc Build-It has three phases: preparation, composition, and editing. During the preparation phase, the writer meets with the technical lead to get a detailed overview of the technology to be documented. The writer then generates a prospective outline for the final document. The outline is granular to the point where each topic has two to three bullet points that suggest what content should appear in the topic.

The composition phase is a single-day event. The writer asks the technical lead to invite three to five key engineers from the team to the event. At the event, the writer first asks the engineers to give a verbal explanation of the technology to get them into a teaching mindset, a mindset that fits documentation. The writer then adjusts the prospective outline based on the explanation provided. Once the Build-It team agrees on the outline, the writer invites the engineers to claim responsibility for the topics in which they feel they have the most expertise. Their task is only to capture the ideas in the words that were spoken during the explanation phase. They then all write simultaneously in the same document. The fact that they can see each other as they produce the sections allows them to align their styles (like level of detail), make cross-references and double check for accuracy.

The writer encourages the engineers to avoid concerns about diction, spelling, punctuation, grammar, or structure. Their contribution to the document is their knowledge. This session normally takes about three hours for relatively simple topics, and up to seven hours for deeper topics.

After the composition phase, the technical writer alone polishes the document. Because the editing process can introduce semantic errors, the writer circulates the edited document for review with the engineering team.

The Build-It method had an enormous impact on productivity. The cost for one Build-It that I ran was an order of magnitude smaller than a traditional documentation approach costing an estimated $1,900 instead of $18,000. Other Build-Its produce similar results, and had ancillary benefits like the identification of subtle bugs in the technology. These discoveries are possible because the event captures the attention of the most knowledgeable people about a topic, who focus closely on the design of the software in the process of creating a document.

Insights. Doing work in clear phases clarified the roles and simplified the whole process. The technical writer created an outline; the outline was then edited after the explanation. Then experts wrote down as much of their knowledge as they could. This knowledge was then cleaned up by the technical writer and reviewed for accuracy. A number of the following Stories have similar mixes of asynchronous and synchronous work.

Story 2: Writing a textbook using IDE, told by Dan. During the summer of 1988, I organized a team of 10 graduate students from the Stanford School of Education to help write a high school algebra textbook, fulfilling a requirement to have experience in designing coursework. The goal was to create a complete textbook with a team in less than 10 weeks. The team wrote this text using IDE.

Of the 10 students, two acted as lead designers and created an initial outline, assigning each chapter to a student author to create. To "write" a chapter, the author had to create a network of nodes in IDE, each linked to other nodes signifying whether the linked-to nodes were abstract concepts, conceptual details, refinements, practice problems or dependencies.

During writing, the authors had to identify any prerequisites that they needed to assume while writing their material. For example, the chapter on Trigonometric functions relied on the Pythagorean theorem to be introduced in an earlier chapter, written by a different author. For that dependency, an explicit prerequisite link would be created to signal that this concept (in one chapter) depended on another concept in another chapter (usually preceding).

The resulting network could then be graph-walked (by following links of a specified type) to produce a single, linear, book-like output, incorporating diagrams and text from each node.

Every Monday, we would meet to review the work done during the preceding week. The team discussed and resolved any overarching issues, and clarified interactions between sections. Since each author could work more-or-less independently (except when they had to edit a node together sitting side-by-side), issues could be resolved fairly easily. The most common issues the team needed to resolve were questions about style, tone, language, and the framing of concepts.

Before each weekly meeting, we would print the graph of all the work done-to-date and post it on the wall for the entire group to see. This poster showed the week-by-week work done by each team member and the links between sections and various subparts. Not only was the graph useful for seeing where issues might arise, but it was also a powerful motivator for other team members to keep up.

During any single week, the authors could work independently in parallel, writing their nodes and creating any substructures as needed. When they discovered new prerequisites, they were responsible for creating those explicit links between the nodes as necessary. Of course, other authors could create new prerequisite links into their territory as well. For the cross-links, authors sometimes chose to work together side-by-side synchronously with one acting as scribe to the discussion.

After eight weeks, the project was completed—all the chapters had complete text, and all the cross-section prerequisite issues had been solved. The printout resulted in a 220-page textbook.

Insights. The book was primarily written in parallel, solo-author sessions, with some short, key moments of simultaneous work when issues arose and resolutions needed to be reached. The book was created over the course of eight weeks, contrasting with the Build-It which had preparation work, one long day of dumping material, and a few days of cleanup. Atlas, O'Reilly (atlas.oreilly.com) and Book-Sprints (www.booksprints.net) support a process similar to this story.

This story is also similar to the one told by Boellstorf et al.,[2] where four authors detail the creation of their coauthored book, written in Google Docs. It recounts initial hand-offs of responsibilities and sections to write. Later they capitalized on the ability to write simultaneously, to find who is writing and where they are writing, and initiate a conversation either by voice or associated chat feature. This was useful, they said, for both quick fixes and encouragement. They also report sessions where one typed while another was dictating, with a third coming along closely behind to make small edits, similar to the upcoming Story 5 about meeting minutes.

Story 3: Creating a committee report using ShrEdit, told by Judy. The computer science department at a major university has an external review every few years. An advisory committee of about eight people from outside the university comes to campus for a few days to hear what the various researchers are doing, review the curriculum, hear statistics on admissions, placement, and so on. Normally, after two-days of presentations, the committee would plan how they were going to write up their report, writing their parts, then reading and commenting on others asynchronously over the next four weeks.

Having found ShrEdit useful in our own work, we organized something like a Doc Build-It, but with some interesting differences. We brought the advisory committee to a special room that was set up with a large number of computers.

We had a single ShrEdit document open with an outline that I had written, which reflected the topics that had been presented over two days prior. All eight people began to write their reactions to and evaluation of the topics. The committee members read the input of others and added their own input. They chose to write wherever they wished, often adding to others' writing, and occasionally having text-based debates.

The committee worked for over an hour, all typing simultaneously. They produced an 11-page document; rough in its style, but full of good content. At the end, one committee member asked, "Where is the cleanup button?" All had a much-needed laugh. The leader of the team volunteered to take this rough draft and make it into good text, taking on a role very much like Ricardo's in the Build-It story. The leader was grateful for the volume of raw material on which to build. This material was far richer than the minutes of a discussion would have been. And when they traveled home, all members of the committee, except the leader, were done.

Insights. Of interest in this story is the fact the writing was not divide-and-conquer like the two previous stories, but rather what we might call a "swarm," for the large number of co-authors involved in this process without a structured process. Everyone contributed to each of the sections at will, sometimes typing very close to another person's current entry. Except for the occasional laugh, the room was silent for an hour with only the sounds of keys clicking softly.

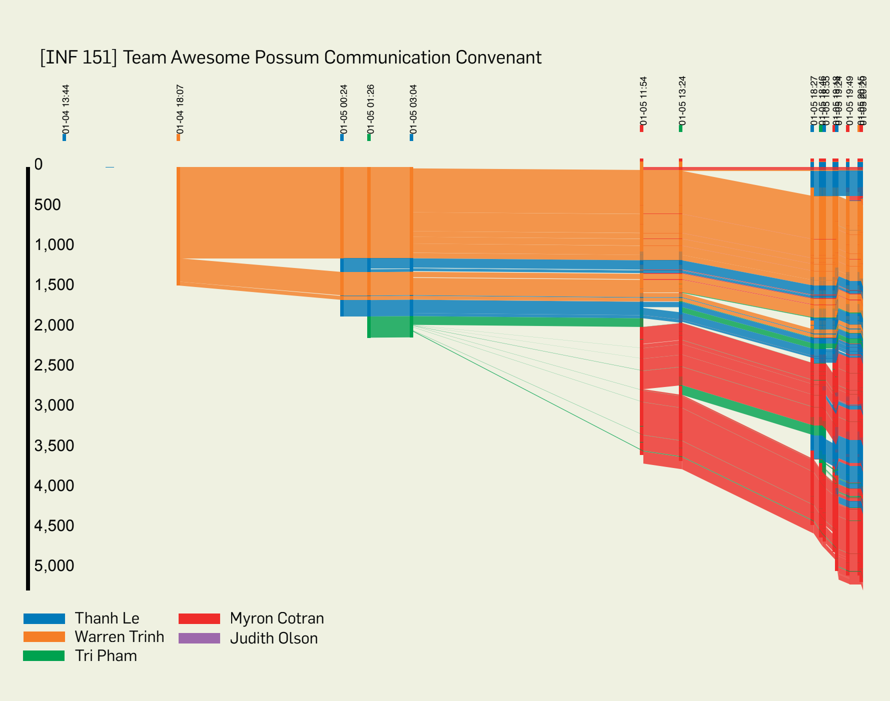
Story 4: Students writing assignments in class using Google Docs, told by Judy. My students in a Project Management course worked in groups to do a small project during the academic quarter. They had to turn in various documents that were common in formal project management practice, such as a business case, a scope statement, and so on. The students were required to use Google Docs and share their final document with me for grading.

With their permission, we analyzed three years worth of the documents—96 in all. We examined their collaborative writing styles and work patterns, and correlated some key features with the quality of the final document. Work patterns were revealed through a tool, DocuViz[11] that shows a picture of the revision history, with slices in time of who wrote what when. The accompanying figure shows the DocuViz timeline visualization of one group's style. Students are represented with different colors, with the size of the stripe indicating length of contribution and different slices indicating when in time they were produced. This team had a lot of simultaneous work near the end.

Some 95% of those documents showed evidence of simultaneous writing, where we defined "simultaneous" as writing activity within seven minutes of the last action without closing and opening the document. One might expect these sessions of simultaneous work to be a "divide and conquer" style. While nearly one-third were, a large number of them showed editing by several people in the same paragraph or even in the same cell of a table.

Insights. We have less detail in this story about how the students managed to create, edit, and vet their work. But traces in the revision histories show not only explicit project management of assigning people to sections, but also freeform editing of their own en-

The DocuViz visualization of a team showing a session of simultaneous work near the end.



tries and, importantly, those of others. There was also a great deal of simultaneous work.

Story 5: Displaying and collaboratively creating meeting minutes using ShrEdit, told by Gary. In addition to using ShrEdit to conduct research,[8] I used it to take minutes during research meetings. In these meetings, the ShrEdit document was shared and open on everyone's workstation, and we often projected it on a large screen at the front of the room. One person, usually me, took the lead, but everyone could edit.

Meeting participants would commonly fix typos and other errors (like correcting the spelling of someone's name) that I or another scribe created while typing rapidly. We often commented that the document looked as if Pac-Man was nibbling along behind the note taker. The shared view of the notes (either from the projected view or the individuals' views on their own computers) helped to maintain the group's focus, and make sure the points were captured correctly. The tool enabled efficient meetings and more accurate notes.

Normally, scribes for a meeting are so busy they cannot contribute easily to the conversation. But when we used ShrEdit, someone else took over the scribe role while the main scribe spoke. This allowed the main scribe to be a full contributor to the meeting.

In one important meeting with corporate sponsors, we projected the meeting minutes on a screen visible to everyone in the room. There were nine agenda items to cover in the afternoon-long meeting. After several hours of talking and taking copious notes, the meeting coordinator noted that we had covered only the first two items with seven still left to go. We had to move on. One of the sponsors said that he was not involved in any of the remaining items, but he had much more to say about the current topic. Consequently, the rest of us went on with the meeting while he continued to write in the appropriate section, hidden from our view. He titled his section "You haven't seen this yet." He wrote his thoughts for an hour, being able to refer to earlier parts of the minutes, while the rest of us finished our agenda. His time was not wasted, and we completed our task. The fact that this was a single document (not an add-on of his thoughts from another document) made his additions easier to read in context.

Insights: Like Stories 1-4, the shared document had a prepared form (the agenda). While three of the previous stories divided the work and gave participants specific assignments to accomplish, in Story 5, one person was the main scribe. Other participants could silently and in parallel, add detail, correct errors, and in this case, continue to contribute while other meeting members went on with agenda items he wasn't involved in. Time was not wasted.

Story 6: Meeting minutes collaboratively and remotely created in Google Docs, told by Gary. The ACM SIGCHI Executive Committee has periodic meetings that last 2–3 days. An agenda is circulated in advance, and modified via email as new items arise. We use Google Docs to take minutes, the document seeded by pasting in the agenda from email. The agenda is often reorganized and modified on the fly.

Like the previous story, one member is designated as the main scribe, but any of us can edit at any time, and when the scribe speaks, someone else temporarily takes over his role.

Occasionally, some members attend parts or even the entire meeting remotely. We would often call remote members for specific items on the agenda, but they observed other parts of the meeting as well. Some even joined *without* having an audio or video connection. In this case, the Google Doc served as a kind of poor man's conferencing tool. The very low rate of change in the Doc works well for a remote participant who can then task switch at appropriate times. We noted also that non-native English speakers benefit from the minutes being a form of "closed captioning."

Often a member of the committee uses PowerPoint slides during their report; these are almost always pasted directly into the shared Google Doc. Occasionally, other kinds of materials are pasted into the Doc, such as links to some web material relevant to what we are discussing.

At another regular research group meeting, the minutes are never deleted but just pushed down, with the new agenda and notes appearing at the top. This archive has proven useful in a number of occasions where someone could resurrect a forgotten "tabled" item.

Insights. This story shares the collaborative note taking with Story 5, but adds the feature of using the emerging document as low bandwidth conferencing and closed captioning.

Story 7: Teaching through noticing things in simultaneous work using Aspects, told by Judy. A former doctoral student of mine, who had graduated recently and was now employed at another university, was scheduled to give a presentation of our joint work at an upcoming conference. I asked the former student to prepare his talk in Aspects' presentation tool, and share it with me.

In a session in which both of us could see the presentation while speaking over the phone, the former student gave his presentation uninterrupted for timing purposes. At the end, I gave general feedback about the timing and order of things. I also suggested including different figures than those he had chosen. While he searched for the new figures and replaced them in the presentation, I made some small formatting changes on a number of slides starting near the end of the deck. When the former student returned, having completed the replacement of figures, he said, "Oh, I see what you're doing." He then changed earlier slides to fit the style of the changes that I was making. We discussed a few more changes in wording. When we were both satisfied with the result, he gave the presentation again without interruption. After two more small changes, we were done.

Gary had a similar experience using Centra screen sharing with the ability to write simultaneously. They went over a draft presentation, discussed it, and made changes in real time. The shared view focused the discussion; simultaneous editing sped the revisions.

Insights. What would have normally taken hours or days of back and forth took one hour. Modeling the types of changes that I wanted enabled the former student to "get it" without discussion. There could have been objections and discussion, but in this case he just copied my style changes without either.

Story 8: Writing this document together with Google Docs: How we did

it. Having discovered that all four of us had stories about writing simultaneously, Ricardo suggested that we write an article on this topic using Build-It. We all happily agreed, wanting both to write this article and to experience Build-It. We scheduled a day for the writing about a month after this initial meeting, with three of us collocated and Dan joining remotely.

We had a videoconference in advance of this day, where we discussed the general framework of the article and the kinds of stories we would use. This was all captured in a Google Doc that we co-wrote. Ricardo created a tentative outline based on this discussion. When we were together, in the first hour we discussed ideas and directions. Initially we were hampered by not having a clear idea of what the article was going to be. Build-It is designed for a divide-and-conquer strategy, not for the stage where we have to discover what we want to say. The outline Ricardo came in with did not capture the discussion we were developing, so we abandoned it.

Eventually we decided the best next step was to simply write up the set of stories that would serve as the raw material. We could read while others wrote. We often read something someone else had written and went back to add or change our own contribution. When everyone was finished, we then read them all in context which then led us to discuss a better organization for the article.

Indeed, in both writing the details and then reading each other's stories, we noted some similarities and differences. We made a working table directly in the document, to compare and contrast the stories. We hoped this would help in both ordering the stories in a sensible way and then supporting the development of the Discussion section.

In the course of creating this table, each of us recognized that we had left something out of the stories. Having noticed these emergent similarities, we combed our own memories for more examples. We named these gathered extra points "epiphenomena."

After about six hours of talking, writing and talking again, we were exhausted, but we had to decide on our next steps. Someone had to do the first pass of ordering the stories, making them a bit more similar in style, and then writing a draft of the Discussion using the table as a guide. Judy volunteered.

The writer encourages the engineers to avoid concerns about diction, spelling, punctuation, grammar, or structure. Their contribution to the document is their knowledge.

This next stage required someone to "own" the document temporarily, without others editing, because a reorganization and a first draft of a discussion required understanding the whole document to make the appropriate connections. When Judy finished, she alerted the others. But we didn't want everyone to edit at once. We first tagged Ricardo, a technical writer, to perform a more fine-grained copyedit. He tried to harmonize the voice and pace of the writing and identify or resolve features of the document that seemed to break the flow.

We then had a videoconference to talk through several important issues, making a list of things to do. We asked Judy and Gary to take the first pass at their parts of this list and then have Ricardo and Dan edit or comment further. Again, noting who was in control was important.

We had not talked about how we were going to manage edits, whether we would just make the changes and count on the revision history to undo things that we disagreed with, use Docs "suggest" mode, which is similar to Word's tracking changes, or use comments to discuss and suggest changes. Judy and Ricardo made changes directly. Dan then used comments to say, for example, "This needs to be spelled out." Dan later clarified that he did not want to edit someone else's work. These examples show that people come to the document with different ideas as to who has the authority or responsibility to change things. As Birnholtz and Ibara[1] found, making changes to others' writing is a social act with consequences having to do with trust and relationships.

The final stage of this collaboration involved putting the Google Doc into Word to fit the required two-column format that this journal requires. From there the editing was entirely hand-off with clear mention of responsibilities and time lines.

Insights. Unlike Build-It, the outline for this article emerged *after* the stories were written down, shared, and discussed. This article also involved using the document as a "holding place" to keep things (like the table) that might help the writing, or material that was eventually deleted. There were times when simultaneous divide-and-conquer was appropriate; there were times when one person had to be in charge while she captured the organization of the emerging article; and there were times when serial hand-off for editing was appropriate. We realized that we should have explicitly discussed whether we would just edit, suggest, or comment for changes.

Discussion

The ability to write simultaneously in a shared document is a powerful advancement in technology. However, the literature has said little about the social process that harnesses the technological advancement of simultaneous writing for real benefit to the users. These stories attempt to shed light on this social process.

In almost all of these stories, someone led the effort by making some sort of structure: the tree in IDE, the outline in ShrEdit and Docs, the agenda in meetings, the presentation draft in Aspects and Centra. The one exception was the writing of this article. We had that discussion after we wrote our stories and read each other's. The structure emerged.

Writing simultaneously offers several benefits, including productivity gains, a deeper sense of satisfaction for time well spent, and practical training by imitation of a collaborator's style. On a tactical level, participants can move quickly toward a quality document because participants can see and emulate what others are doing. As people join in writing, they can view recent work in order to make their contributions fit the overall vision. The ability to work simultaneously on meeting minutes has benefits beyond the recording and correcting the content. Everyone was "on the same page."

Of course, not all collaborations may benefit from simultaneous work. There are sensitivities when someone changes your writing. There are sensitivities when others can see your process of writing (for example, if you are slow or a bad speller). Some may find it distracting to see the edits of others while they are writing or editing. Working simultaneously is not appropriate if you mistrust your colleagues, either because it is a competitive atmosphere or their expertise and abilities are not on par with your own. In some competitive cultures, there can be sabotage. In addition, many of the sessions of simultaneous work that we outline were for rough work, not for the final draft (although Story 7 contained the collaborative reformatting of a final presentation).

When we examine the stories, it is striking that there is not more diversity in the experiences, but rather just a few different patterns of writing together.

And, there can be technical difficulties, such as edit wars because with direct editing, one does not see what something was changed from. Also, if two people are in exactly the same point with one adding and one deleting, it can get very confusing. And, to write simultaneously, one has to be on a network, having to rely on server connectivity. In addition, if using a service in the cloud, some may have concerns about privacy.

Patterns of using simultaneous writing. When we examine the stories, it is striking that there was not more diversity in the experiences, but rather just a few different patterns of writing together. The stories cluster into two sets: Four patterns of simultaneous work and two patterns of accompanying asynchronous hand-offs.

The first four stories and the account of writing this article employ a simultaneous *divide-and-conquer* strategy. At some points in the creation of the document, all co-authors wrote at the same time. Most often they wrote in different sections of the document.

Stories 5 and 6 employ a main scribe, with a second or third scribe involved either to immediately take over when the primary scribe speaks or to do ancillary additions or edits. We call this the *rotating scribe* pattern.

Story 5 employs a *branching* pattern, where when one person is not involved in the immediate conversation, they use the time productively to write more for others to read later. In essence, the one person is creating a new branch in the minutes, while the others proceed. This pattern is a variant of the divide-and-conquer.

The fourth pattern is exemplified in Story 3, what we call the *swarm*. In this pattern, everyone is in the document writing their parts, reading other's and commenting or correcting them. No one is assigned a section; they all are responsible for the whole document. This is also similar to the teaching stories, where the "teacher" exemplifies what the "student" is to emulate, and they collectively finish the document.

The fifth pattern, the *cleanup*, is a solo kind of work that appears in many of the stories. For example, in our own writing, there was one big simultaneous session and then Judy took over to both re-organize the stories into clusters and write the text that represented the discussion captured in the table. In the committee report, the chair took on the role of cleaning up the rough text and synthesizing content. In BuildIt, Ricardo takes on the responsibility of cleaning up the text and coordinating tone, turning rough text into good prose.

The sixth pattern, again often represented in many of the stories, is the *hand-off*, where different authors are "in charge" for periods of time for reorganization or fact-checking and simpler editing. This often accompanied sessions of simultaneous work.

We also note that collocation for synchronous writing was important but not necessary. Collocation provides immediate access to other participants for things such as clarification, seeing people's expressions, indicating that one wanted a turn to speak, and so on. However, many of the stories had successful contributions from remote people, even without audio or video connectivity. In one case, remote participants *only* saw the evolving Google Doc. For some purposes, this level of participation was sufficient.

Epiphenomena. Late in our discussion of the similarities and differences among the stories, we noted some epiphenomena—unusual behavior generated by the fact the work was being created synchronously in view of all.

One epiphenomenon involved *humor*, often an important social component of intense work. For example, as the scribe in meetings where the minutes are projected, Gary would often type slightly snide comments about what was being said and quickly erase them. For example, if someone was talking too long, Gary would write in big letters but out of the speaker's view, "Is it time for lunch yet?" and then quickly erase it.

A student in one Project Management team, deep in discussion in a long simultaneous-editing session about their client, pasted in a funny picture of a grizzly man wearing a bear hat and the text "this guy!" The picture was in the document for about a minute then deleted. The Visiting Committee read emerging debates inside the report, eliciting giggles occasionally. At the end, one member of the committee also remarked for humor, "Where is the Clean-It-Up button?" The collective laugh was one of appreciation and relief that the session was over.

A second epiphenomenon we noted is that visibility is motivating. In using IDE to write the textbook, the visible presentation of progress on the book motivated people. Similarly, while the document is live and worked on simultaneously, one can see where the activity is and be motivated to read carefully and talk about any arising issues either through text chat or voice conversation. When there is visible activity, people feel compelled to focus on the document being created. One attends to "seismic activity." We believe the student who pasted in a funny picture during a simultaneous working session used the humor to induce motivation to keep working. One student in another team pasted in a picture of a shovel and wrote "Work Hard!"

Conclusion

Writing simultaneously is an extremely powerful capability, now widely available in commercial software. It is often successfully mixed with some hand-off writing and some sessions where one person takes charge to integrate the material and voice. But technology alone does not make enhanced productivity and satisfaction, people do. What we offer here are a number of stories and commentary on what makes this kind of writing so powerful.

We follow the stories with six patterns of writing when simultaneous work is possible. Team members now need to plan the style of work (some of which will include simultaneous writing) that would fit the kinds of goals at hand. As outlined by Glushko[3] some of the overall collaborations will be preplanned (what he calls hierarchical collaboration), co-developed by the authors (what he calls consensus collaboration), or a free-for-all (what he calls open collaboration). Each approach has its strengths and weaknesses, so choosing appropriately for the task at hand is important. And, as mentioned earlier, not all teams and occasions will benefit; cooperation and trust are essential.

We have provided a set of examples where we have worked in new ways that are very productive, and therefore satisfying. The next step is for you to analyze the collaborative writing situations you are in day-by-day and craft a method that best suits. The possibilities are very rich.

Acknowledgments. Portions of this work were supported by National Science Foundation grant ACI-1322304 and a Google Focused Research Award to Judy and Gary Olson. Helpful comments on an earlier draft were provided by Tom Boellstorf, Bonnie Nardi, and Bob Glushko along with anonymous reviewers.

References

1. Birnholtz, J. and Ibara, S. Tracking changes in collaborative writing: Edits, visibility, and group maintenance. In *Proceedings of CSCW'12*, 2012, 809–818.
2. Boellstorff, T., Nardi, B., Pearce, C., and Taylor, T.L. Words with friends: Writing Collaboratively Online. *Interactions 20*, 5 (Sept.-Oct. 2013), 58-61.
3. Glushko, R. Collaborative authoring, evolution, and personalization for a 'Transdisciplinary' textbook. In *Proceedings of OpenSim 2015*.
4. Group Technologies. Aspects: The first simultaneous conference software for the Macintosh, Version 1. Manual, (1991). Group Technologies, Inc., Arlington, VA.
5. Halasz, F.G., Moran, T.P., and Trigg, R.H. Notecards in a nutshell. In *Proceedings of the SIGCHI-GI Conference on Human Factors in Computing Systems and Graphics Interface* (1987), 45–52.
6. Irish, P.M., and Trigg, R.H. Supporting collaboration in hypermedia: Issues and experiences. *J. American Society for Information Science 40*, 3 (1989), 192–199.
7. McGuffin, L. and Olson, G.M. ShrEdit: A shared electronic workspace. CSMIL Tech. Report, 1992. The University of Michigan, Ann Arbor, MI.
8. Olson, J.S., Olson, G.M, Storrosten, M. and Carter, M. Groupwork Close Up: A Comparison of the Group Design Process With and Without a Simple Group Editor. *ACM Trans. on Information Systems 11*, 4 (1993), 321–348.
9. Olson, J.S., Wang, D., Zhang, J. and Olson, G.M. How people write together now. *Trans. Computer-Human Interaction 24*, 1 (2017), 1–40.
10. Russell, D.M., Burton, R.R., Jordan, D.S., Jensen, A-M., Rogers, R.A., and Cohen, J.R. Creating instruction with IDE: Tools for instructional designers. *Intelligent Tutoring Media 1*,1 (2009), 3–16.
11. Wang, D., Olson, J.S., Zhang, J., Nguyen, T. and Olson, G.M. DocuViz: Visualizing Collaborative Writing. In *Proceedings of the Conference on Human Factors in Computing Systems*, (2015), 1865–1874.

Ricardo Olenewa is a technical writer living in Waterloo, Ontario, Canada.

Gary M. Olson (gary.olson@uci.edu) is Professor Emeritus of Informatics at the University of California at Irvine.

Judith S. Olson (jolson@uci.edu) is Professor Emerita of Informatics at the University of California at Irvine.

Daniel M. Russell (drussell@google.com) is a senior research scientist at Google, Mountain View, CA.

DOI:10.1145/2911981

Rounding errors are usually avoidable, and sometimes we can afford to avoid them.

BY HANS-J. BOEHM

Small-Data Computing: Correct Calculator Arithmetic

COMPUTERS COMMONLY PERFORM numerical computations using floating point arithmetic,[a] typically representing numbers as specified by the IEEE 754 standard. Numbers are represented as `m x b`^{e}, where `b` is base, `m` is a fixed bit length length fraction (*mantissa*), with an implicit "decimal point" on the left, and `e` is an exponent. For conventional IEEE "double precision" floating point, the base `b` is 2, and the mantissa `m` is 53 bits (approximately 16 decimal digits) long. For a hardware calculator, we might use `b = 10`, with a 12-digit mantissa.[b]

Floating-point representations are used pervasively, from large-scale scientific computing problems down to pocket calculators. They provide a great time-honored compromise between speed of computation and sufficient accuracy to usually provide meaningful results. A 53-bit mantissa can often provide 10 to 15 decimal digits of accuracy in the final result, and modern processors can often perform more than one floating-point operation per cycle.

But conventional floating point arithmetic remains a compromise. The results can be computed quickly, but they are only usually precise enough to be meaningful. Usually the precision loss from rounding to 53 bits is not noticeable, because we are usually computing on measured physical quantities that are far less accurate to start with, and usually well-designed algorithms do not compound these incoming measurement errors too much. But none of those "usually" qualifiers can be dropped, and algorithms are not always well designed.

Most of us are familiar with some of the programming hazards of floating point. We may have observed, for example, that the loop

```
for (x = 0.0; x != 10.0; x += 0.1) { … }
```

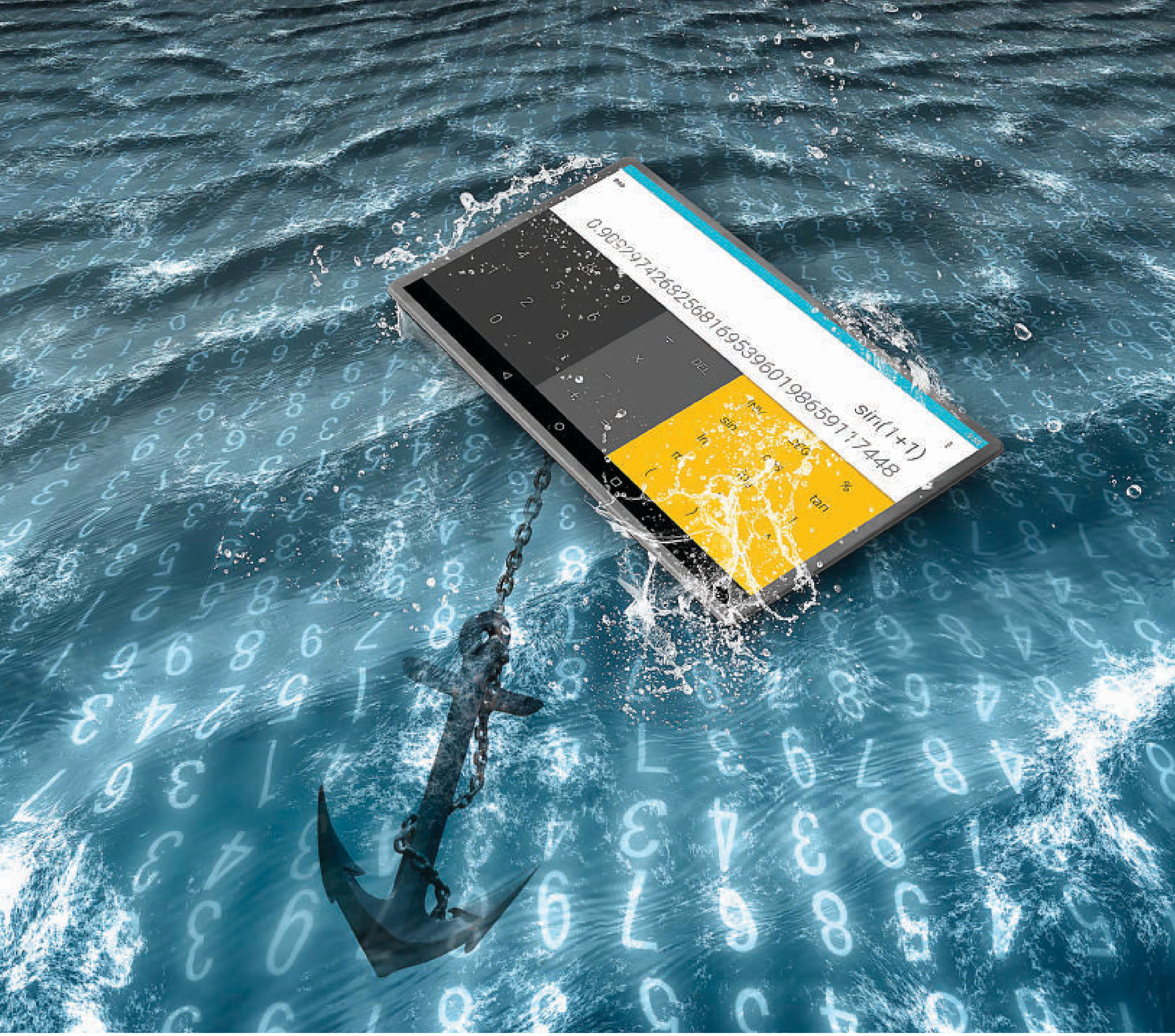
usually fails to terminate. But we are willing to deal with these issues, and write more careful code, in order to get high performance numerical computation.

But sometimes performance, at least in the conventional sense, really doesn't matter. Calculators, which normally target expressions with few operations, are probably the canonical example for this category of applications. That is particularly true when the calculator is really an application running on a smartphone with four 2GHz processor cores. This is also an environment in which users are unlikely to think much about formulating algorithms to optimize floating point precision properties.

Even for calculators, the hazards of floating point extend to more than a few digits off in the last place. For example, if we try to compute:

[a] Goldberg, D. What every computer scientist should know about floatingpoint arithmetic. *ACM Computing Surveys 23*, 1 (1991), 5–48.

[b] Cochran, D.S. Internal programming of the 9100A Calculator. *HP Journal*, Sept. 1968.



$$\sqrt{1 + 10^{-16} - 1 - 10^{-16}}$$

which is clearly equivalent to √0, on any standard calculator, the result is just an error message. The problem is that $1 + 10^{-16}$ is rounded to 1. When we subtract 1, we get 0 instead of 10^{-16}. This is commonly known as "catastrophic cancellation:" We are subtracting two nearly equal numbers, effectively magnifying the input error, yielding a result with few or no meaningful digits. When we then subtract 10^{-16}, the result is a negative number, which can now be represented accurately. Taking the square root of a negative number produces the error. For a more interesting and subtle example along these lines, see the sidebar entitled "A Fixed-Precision Fail."

IMAGE BY ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES

There are other cases for which we do get correct answers, but the first 16 digits fail to expose the interesting properties of the result. We may want to see when the decimal expansion of a rational number repeats. Or we may want to see how close Ramanujan's constant ($e^{\pi\sqrt{163}}$) is to an integer. These tend to be "mathematical" rather than "physical" problems. But we suspect a significant fraction of calculator use is in schools for just that purpose.

The Space Beyond Machine Floating Point

Perhaps the most serious problem with conventional calculator arithmetic is that it has trained us not to even attempt certain problems.

Every calculus class teaches us that we can approximate derivatives with finite differences. We can, usually very accurately, approximate $f'(x)$ by $(f(x + h) - f(x))/h$, with a sufficiently small h. For example, since the derivative of e^x is e^x, we should expect that $(e^{1+h} - e^1)/h$ evaluates to a very good approximation of e, if we use $h = 10^{-1000}$.

This of course does not work on normal calculators. The expressions e^{1+h} and e^1 agree to far more digits than the evaluation precision, and the numerator evaluates to zero, yielding a "derivative" of 0 rather than e.

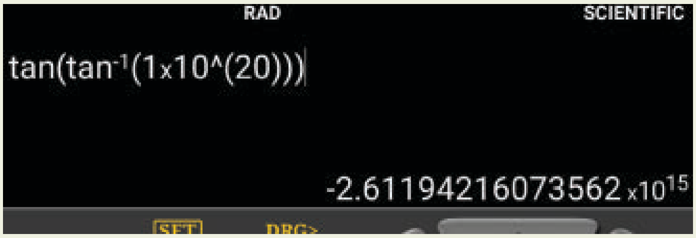
In fact, the idea of limited precision seems to be sufficiently drilled into us that it occurs to few people to even try something like this. And

A Fixed-Precision Fail

Although conventional calculators usually provide plenty of precision for most purposes, they are prone to occasional misbehavior on surprisingly simple problems. These are more likely to be problems encountered by high school or college students experimenting with mathematical laws, than they are real engineering problems.

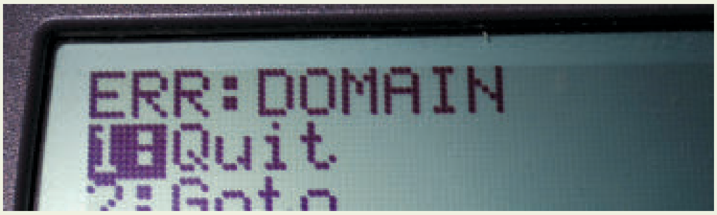
One particularly small example involves the trigonometric tangent (`tan`) and arctangent (`tan`$^{-1}$) functions. The expression `tan(tan`$^{-1}$`(x))` maps a slope *x* to the corresponding angle and back, and should clearly always give us back *x*. But evaluating this expression correctly for large x tends to be quite tricky, since `tan`$^{-1}$ of a large argument produces a result very close to π/2 radians (or 90 degrees), which is a singularity for the tan function.

On software-based calculators (Android or Web), a result something like the one shown here is common for `tan(tan-1(10`^{20}`))`.



```
tan(tan⁻¹(1x10^(20)))
-2.61194216073562 x10¹⁵
```

On old-fashioned hardware calculators we tried, the arguably better answer shown below is more common.



```
ERR:DOMAIN
1:Quit
2:Goto
```

The standard Microsoft Windows calculator we tried uses more precision, and hence fails only with even larger values of *x*.

people often seem surprised when we suggest it.

For numerical differentiation with machine floating point, there is a subtle trade-off in the choice of h, which is likely to be well beyond the expertise of a high school student trying to check a formula on a calculator. And yet there is no reason calculations like this shouldn't just work, even with $h = 10^{-1000}$. The sidebar entitled "Derivatives on a Calculator" pushes this example a bit further.

Our Starting Point

The Android Open Source Project has always included a relatively simple calculator application. This is the default calculator on Pixel phones and many other Andriod devices. Historically some other third-party Android calculators have also extended the same source code base. This calculator is designed to be as simple as possible, targeting primarily nontechnical users rather than engineers. It has always offered "scientific calculator" functionality, with trigonometric functions, among others. But the emphasis has been on simple use cases and conformance to Android user interface guidelines.

In versions prior to Android 6.0 Marshmallow, the calculator internally used the "arity" expression evaluation library.[c] The calculator uses this library to evaluate traditional infix expressions. Conventional syntax is mildly extended to allow dropping of trailing parentheses and a few other shortcuts. The actual evaluation is performed using double precision machine floating point.

The calculator rounded the result to a smaller number of decimal digits than the 16 provided by the underlying arithmetic. This reduced the probability that a number with a finite decimal expansion like 12.34, but an infinite binary expansion, would be displayed as 12.399999999. But there was an unavoidable tension between not dropping enough digits and generating unpleasant representations, and dropping too many. The latter would either introduce additional error for intermediate results, or it would force the calculator to display a result significantly different from its internal representation. Both could produce unpleasant surprises. A Web search for "Android Calculator bug" shows some of the results. The nature of the complaints also confirms that most users are not happy to tolerate the kind of floating-point issues that a numerical analyst would expect.

Accurate Answers From a Calculator

Our goal was to replace the arithmetic evaluation engine in the Android Calculator with one that was not subject to floating point-rounding errors. We wanted to ensure at a minimum that the displayed final answer was never off by one or more in the final displayed digit.

This can be done using "constructive real arithmetic."[d,e] Rather than computing a fixed number of digits for each intermediate result, each subexpression is computed to whatever precision is needed to ensure sufficient accuracy of the final result.

Let's say we want to compute $\pi + {}^{1}\!/_{3}$, and the calculator display has 10 digits. We would compute both π and $^{1}\!/_{3}$ to 11 digits each, add them, and round the result to 10 digits. Since π and $^{1}\!/_{3}$ were accurate to within 1 in the 11th digit, and rounding adds an error of at most 5 in the 11th digit, the result is guaranteed accurate to less than 1 in the 10th digit, which was our goal.

Other operations are somewhat more complex to implement. Multiplication to 10 digits beyond the decimal point might involve evaluating one argument to five digits. If that is zero, we

[c] Mihai Preda, https://code.google.com/p/arity-calculator/

[d] E. Bishop, and D. Bridges. *Constructive Analysis.* Springer Science & Business Media, 1985.

[e] Boehm, HJ., and Cartwright, R. Exact real arithmetic: Formulating real numbers as functions. Rice University, Department of CS, 1988.

evaluate the other argument to five digits. If both are zero, zero is an acceptable answer. Once we have a nonzero argument, we can get a reasonably tight bound on the number of digits needed for the other argument, and use that to reevaluate the initial nonzero argument to the correct precision.

Some functions, such as square roots, can be easily evaluated to any required precision using Newton iteration. Common transcendental functions can be evaluated using Taylor series, taking care to evaluate sufficiently many terms to sufficient precision to guarantee the 1-digit-in-the-last-place error bound.

A number of detailed representations for the constructive reals have been explored.[f,g,h,i]

We started with an existing Java library,[j] mildly enhancing it as needed. The library represents real numbers as class CR Java objects with an `appr()` method. A call to `appr(`n`)`, where n is typically negative, produces an approximation accurate to 2^n. The actual result returned is implicitly scaled (multiplied) by 2^{-n}, so that it can be represented as an integer. For example, if `THREE` is the constructive real representation of 3, then `THREE.appr(−3)` would yield 24, that is, 3 multiplied by 2^3 or 8. That would be the only acceptable answer, since the result always has an error of < 1.

To add two numbers in this representation, we produce an instance of a subclass of CR, implemented as:

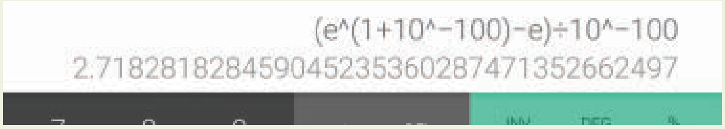
```
class add_CR extends CR {
   CR op1; CR op2;
   ...
   protected BigInteger
   appr(int p) {
      return scale(op1.appr(p-2).
       add(op2.appr(p-2)), -2);
   }
}
```

Here `scale( ..., n )` multiplies by 2^n and rounds to the nearest integer, ensuring a final rounding error of $\leq \frac{1}{2}$. The arguments are evaluated to two additional bits, ensuring that each contributes an error of $< \frac{1}{4}$.

Derivatives on a Calculator

We explore tiny finite differences to approximate derivatives a bit further. We continue to use the exponential function as an easy-to-type and easy-to-check example.

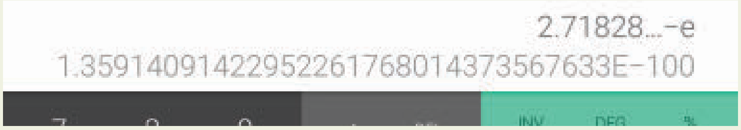
The figure here illustrates the result of computing the derivative of e^x at 1, by computing $(e^{1+h} - e^1)/h$, with $h = 10^{-100}$:



(e^(1+10^−100)−e)÷10^−100

2.718281828459045235360287471352662497

As expected, the result is unsurprising to anyone untrained in floating point arithmetic. It looks exactly like e. We can now easily ask what error resulted from approximating the derivative as a finite difference, rather than computing its limit. The following figure shows what we get if we take the preceding result and subtract e:



2.71828...−e

1.35914091422952261768014373567633E−100

A trained numerical analyst might cringe at the observation that this computation involves a *second* catastrophic cancellation in a row. The astute reader might also notice that this is itself astonishingly close to ½ e x 10^{-100}. We leave it to the interested reader to go for three catastrophic cancellations in a row and calculate the difference.

The real implementation caches the highest precision prior evaluation, so reevaluating an expression to fewer digits of precision is essentially free.

Calculators based on constructive real arithmetic are not new. The library we use as a basis contains a basic Java applet calculator. WolframAlpha also appears to use a technique along these lines.[k] However, we had two additional, previously unsatisfied, goals:

1. It was essential that the calculator remain usable as a general-purpose tool, for example, for balancing checkbooks and calculating tips, and for mathematically unsophisticated users. We wanted behavior universally better than machine floating point.

2. We want an intuitive way to present numbers with nonterminating decimal representations as infinite objects, as opposed to explicitly entering a result precision.

We now focus on these issues.

Scrollable Results

Since we must be able to produce answers to arbitrary precision, we can also let the user specify how much precision she wants, and use that to drive the evaluation. In our calculator, the user specifies the requested precision by scrolling the result, as one would expect with a primarily touch-based user interface.[l]

In order to preserve the illusion of an infinite result as much as possible, we precompute a higher precision result in the background, as soon as we have displayed about $^4/_5$ of the digits computed. The number of additional digits computed each time is a bit more than $^1/_5$ of the number we have computed so far, so we recompute in larger chunks the further the user scrolls, and the more expensive the computations become. This typically succeeds in hiding scrolling latency for a few thousand digits, even if the user resorts to "fling" gestures to scroll quickly.[m]

[f] Aberth, O. A precise numerical analysis program. *Commun. ACM 17*, 9 (Sept. 1974), 509–513.

[g] Ménissier Morain, V. Arbitrary precision real arithmetic: Design and algorithms. *J. Logic and Algebraic Programming 64*, 1 (2005), 13–39.

[h] Vuillemin, J.E. Exact real computer arithmetic with continued fractions. *IEEE Trans. Computers 39*, 8 (1990), 1087–1105.

[i] Lee, Jr, V.A. and Boehm, H-J. Optimizing programs over the constructive reals. ACM, 1990.

[j] Boehm, H.J. The constructive reals as a Java library. J. *Logic and Algebraic Programming 64*, 1 (2005), 3–11.

[k] It also appears to use a few, but not all, of the techniques described in this article. We could not find a detailed description.

[l] A brief demonstration video can be found at https://vimeo.com/219144100.

[m] The computation cost of a Taylor series expansion is typically around $O(n^{2.6})$ for n digits (O(n) terms, each of which requires a constant number of Karatsuba multiplications on O(n) digits), this eventually falls behind a constant speed scroll, even if the reevaluation frequency decreases linearly.

Indicating position. We would like the user to be able to see at a glance which part of the result is currently being displayed.

Conventional calculators solve the vaguely similar problem of displaying very large or very small numbers by using scientific notation. We use the same approach for the initial display.[n] If the user enters “1÷3x10ˆ20”, computing $^1/_3$ times 10 to the 20th power, the result may be displayed as 3.3333333333E19. In this version of scientific notation, the decimal point is always displayed immediately to the right of the most significant digit.

Once the decimal point is scrolled off the display, this style of scientific notation is not helpful; it essentially tells us where the decimal point is relative to the most significant digit, but the most significant digit is no longer visible. We address this by switching to a different variant of scientific notation, in which we interpret the displayed digits as a whole number, with an implied decimal point on the right. Instead of displaying 3.3333333333E19, we hypothetically could display 33333333333E9 or 33333333333 times 10^9. In fact, we use this format only when the normal scientific notation decimal point would not be visible. If we had scrolled the above result two digits to the left, we would in fact be seeing ...33333333333E7. This tells us the displayed result is very close to a whole number ending in 33333333333 times 10^7. The two forms of scientific notation are easily distinguishable by the presence or absence of a decimal point, and the ellipsis character at the beginning.

Perhaps the most serious problem with conventional calculator arithmetic is that it has trained us not to even attempt certain problems.

Rounding vs. scrolling. Normally we expect calculators to try to round to the nearest displayable result. If the actual computed result were 0.66666666666667, and we could only display 10 digits, we would expect a result display of, for example 0.666666667, rather than 0.666666666. For us, this would have the disadvantage that when we scrolled the result left to see more digits, the “7” on the right would change to a “6”. That would be mildly unfortunate. It would be somewhat worse if the actual result were exactly 0.99999999999, and we could only display 10 characters at a time, we would see an initial display of 1.00000000. As we scroll to see more digits, we would successively see ...000000E-6, then ...000000E-7, and so on until we get to ...00000E-10, but then suddenly ...99999E-11. If we scroll back, the screen would again show zeroes. We decided this would be excessively confusing, and thus try to truncate toward zero rather than rounding.

It is still possible for previously displayed digits to change as we are scrolling. But we always compute a number of digits more than we actually need, so this is exceedingly unlikely.

Since our goal is an error of strictly less than one in the last displayed digit, we will never, for example, display an answer of exactly 2 as 1.9999999999. That would involve an error of exactly one in the last place, which is too much for us.

It turns out there is exactly one case in which the display switches between 9s and 0s: A long but finite sequence of 9s (more than 20) in the true result can initially be displayed as a slightly larger number ending in 0s. As we scroll, the 0s turn into 9s. When we immediately scroll back, the number remains displayed as 9s, since the calculator caches the best-known result (though not currently across restarts).

We prevent 9s from turning into 0s during scrolling. If we generate a result ending in 9s, our error bound implies that the true result is strictly less (in absolute value) than the value (ending in 0s) we would get by incrementing the last displayed digit. Thus we can never be forced back to generating zeros and explicitly ensure that we always continue to generate 9s, and 9s never turn into 0s.

Coping with Undecidability

The calculator essentially represents a number as a program for computing approximations. This representation has many nice properties, like never resulting in the display of incorrect results. It has one inherent weakness: Exact equality of two numbers is fundamentally undecidable. We can compute more and more digits of both numbers, and if they ever differ by more than one in the last

[n] Numbers that differ from zero by less than 10^{-320} may be displayed as 0.0000000000. See section Coping with Undecidability.

computed digit, we know they are not equal. But if the two numbers were in fact the same, this process would go on forever.

This still improves on floating-point arithmetic—equality is easily decidable, but tells us even less about equality of the true mathematical real numbers approximated by the floating point values.

This undecidability of equality does create some interesting issues. If we divide a number by x, the calculator will compute more and more digits of x until it finds some nonzero ones. If x was, in fact, exactly zero, this process will continue forever.

We deal with this problem using two complementary techniques:

1. We always run numeric computations in the background, where they will not interfere with user interactions, just in case they take a long time. If they do take a really long time, we time them out and inform the user that the computation has been aborted. This is unlikely to happen by accident, unless the user entered an ill-defined mathematical expression, like a division by zero.

2. As we will see, in many cases we use an additional number representation that does allow us to determine that a number is exactly zero. Although this easily handles most cases, it is not foolproof. If the user enters "1÷0" we immediately detect the division by zero. If the user enters "1÷(π^2÷π−π)" we time out.

Zeros Further Than the Eye Can See

Prototypes of our calculator, like mathematicians, treated all computed results as infinite objects, with infinitely many digits to scroll through. If the actual computation happened to be 2 − 1, the result was initially displayed as 1.00000000, and the user could keep scrolling through as many thousands of zeroes to the right of that as he desired. Although mathematically sound, this proved unpopular for several good reasons, the first one probably more serious than the others:

1. If we computed $1.23 + $7.89, the result would show up as 9.1200000000 or the like, which is unexpected and confusing.

2. Many users consider the result of 1+2 to be a finite number, and find it confusing to be able to scroll through lots of zeros on the right.

3. Since the calculator could not ever tell that a number was not going to be scrolled, it could not treat any result as short enough to allow the use of a larger font.

These problems were largely addressed by evaluating expression to not just a constructive real number, but also to a rational number represented as a (numerator, denominator) pair. The latter is unavailable if the computation involved an irrational number, or the rational representation is too large.

This allows us to tell whether a result has a finite decimal representation, and if so, how many digits there are to the right of the decimal point. We simply look at the fraction in lowest terms. If the denominator has a prime factor other than 2 or 5, the decimal expansion is clearly infinite; no number of multiplications by 10 can turn the fraction into an integer. Otherwise the denominator can be factored as 2^n5^m and the number of nonzero digits to the right of the decimal point is `max(`m,n`)`.

If the expansion is finite, we prevent scrolling past that point. We also prevent scrolling through a large number of trailing zeroes to the left of the decimal point. This often leaves us with a short nonscrollable result, for which we can use a larger font. Unlike the floating-point case, such short, large font results are always exact, and never attributable to a number that was merely close to a short decimal representation.

This is, however, fallible in the other direction. For example, we do not compute a rational representation for 1÷(π^2÷π−π), and hence it is still possible to scroll through as many zeros of that result as you like.

This underlying fractional representation of the result is also used to directly detect, for example, division by zero, making it much less likely that a casual user will ever see a timeout.

Looking Back

The calculator described here is available through Google Play Store, and is also the default calculator distributed with Android 6.0 Marshmallow and later.

Initial reviews of the calculator liked several unrelated UI and functionality changes, but failed to notice the change in arithmetic.[o] We were apparently successful in having the accuracy guarantees not interfere with normal use.

The calculator now exposes the rational representation to the user when it is available. That has turned out to be a useful feature on its own, though it was motivated by other considerations.

Feedback has been quite positive. But it, together with our own experience, has also suggested some improvements:

- Scrolling results have generated far more interest than the much more subtle precision improvements. The latter seem to have been recognized only by an absence of bug reports. As a result, performance of a different kind actually does matter: Users did notice sluggishness when scrolling through 30,000 digits of π. And we subsequently switched to the better-performing Gauss-Legendre algorithm for π.
- The semantics of calculator expressions are more subtle and controversial than we had realized. Is 50+10% equal to 50.1 or 55? If the latter, what's 50x5%? If 2π is a valid expression, what about π2?
- The most recent versions of our calculator explicitly track rational multiples of π and some other common irrational constants. This allows us to compute a rational result for `sin`(π/6) in radian mode, as we already did for `sin`(30°).

Acknowledgments

The calculator UI design and implementation of course relied on contributions from several others, most notably Justin Klaassen.

[o] For example, see http://www.androidpolice.com/2015/06/12/android-m-feature-spotlight-the-stock-calculator-app-has-been-overhauled-no-longer-uses-floating-point-arithmetic/.

Hans-J. Boehm (boehm@acm.org) is a software engineer at Google, Palo Alto, CA.

DOI:10.1145/3104032

Turing's machines of 1936 were a purely mathematical notion, not an exploration of possible blueprints for physical calculators.

BY LEO CORRY

Turing's Pre-War Analog Computers: The Fatherhood of the Modern Computer Revisited

ALAN TURING IS often praised as the foremost figure in the historical process that led to the rise of the modern electronic computer. Particular attention has been devoted to the purported connection between a "Universal Turing Machine" (UTM), as introduced in Turing's article of 1936,[27] and the design and implementation in the mid-1940s of the first stored-program computers, with particular emphasis on the respective proposals of John von Neumann for the EDVAC[30] and of Turing himself for the ACE.[26]

In some recent accounts, von Neumann's and Turing's proposals (and the machines built on them) are unambiguously described as direct implementations of a UTM, as defined in 1936. Most noticeable in this regard are the writings of Jack Copeland and his collaborators, as stated in the following example:

"What Turing described in 1936 was not an abstract mathematical notion but a solid three-dimensional machine (containing, as he said, wheels, levers, and paper tape); and the cardinal problem in electronic computing's pioneering years, taken on by both 'Proposed Electronic Calculator' and the 'First Draft' was just this: How best to build a practical electronic form of the UTM?"[9]

Similar is the following by Andrew Hodges:

"[The] essential point of the stored-program computer is that it is built to implement a logical idea, Turing's idea: the universal Turing machine of 1936."[18]

This statement is of particular interest because, in his authoritative biography[21] of Turing (first published 1983), Hodges typically follows a much more nuanced and careful approach to this entire issue. For instance, when referring to a mocking 1936 comment by David Champernowne, a friend of Turing, to the effect that the universal machine would require the Albert Hall to house its construction, Hodges commented that this "was fair comment on Alan's design in 'Computable Numbers' for if he had any thoughts of making it a prac-

» key insights

- There is no straightforward, let alone deterministic, historical path leading from Turing's 1936 ideas on the Universal Machine to the first stored-program electronic computers of the mid-1940s.
- Turing's own pre-war ideas on the Universal Machine were not intended as a possible blueprint for the actual construction of physical automatic calculating machines.
- Turing's personal interaction with von Neumann while at Princeton had little impact, if at all, on the later involvement of both men on the design and construction of the early stored-program computers, beginning in the mid-1940s.

Alan Turing finishing a race, 1946.

PHOTO: NATIONAL PHYSICAL LABORATORY © CROWN COPYRIGHT/SCIENCE PHOTO LIBRARY

tical proposition they did not show in the paper."[21] Or, even more cautiously, in the following:

"Did [Turing] think in terms of constructing a universal machine at this stage? There is not a shred of direct evidence, nor was the design as described in his paper in any way influenced by practical considerations ... My own belief is that the 'interest' [in building an actual machine] may have been at the back of his mind all the time after 1936, and quite possibly motivated some of his eagerness to learn about engineering techniques. But as he never said or wrote anything to this effect, the question must be left to tantalize the imagination."[21]

Discussions of this issue tend to be based on retrospective accounts, sometimes even on hearsay. The most-often quoted one comes from Max Newman, who had been Turing's teacher and mentor back in the early Cambridge days and, later, became a leading figure in the rise of the modern electronic computer, sometimes collaborating with Turing. In an obituary published in 1954, he wrote:

"The description that [Turing] gave of a 'universal' computing machine was entirely theoretical in purpose, but Turing's strong interest in all kinds of practical experiment made him even then interested in the possibility of actually constructing a machine on these lines."[6]

This and other similar testimonies have been cited repeatedly as solid historical evidence but are invariably vague and unsupported.[a] Similar is the case with the anecdotes about the purported early influence of Turing's paper on von Neumann; see, for example, Hodges.[21]

This article is intended as a further contribution to the historical ongoing debates about the actual role of Turing in the history of the modern electronic computer and, in particular, the putative connection between the UTM and the stored-program computer. I contend that in order to achieve a complete and balanced historical picture, one must explicitly abandon the idea of a straightforward (let alone necessary) transition from the mathematical idea of 1936 to the physical machine (or even the design of that machine) in 1945. More specifically, by exploring the details of Turing's pre-war involvement with various fields of mathematics, both at Cambridge and at Princeton, and with the actual construction of two calculating machines, I claim that to the extent that early stored-program computers of the mid-1940s can be seen as physically embodying ideas discussed in the 1936 paper, "Computable Numbers," this is mostly a result of hindsight and says little about Turing's ideas before the war.

The purported connection I call into question involves, in the first place, a technical claim, namely that the UTM as defined in 1936 comprises a representative mathematical model of the stored-program electronic computers of the late-1940s. Turing, in 1947, said, for example, that digital computing machines (such as the ACE) "are in fact practical versions of the universal machine."[6] But the full validity of this technical claim is debatable in various ways. For one, a stored-program machine is only one way to construct a practical realization of a UTM. For another, even when the connection was mentioned in relation to the new machines, reference was to re-cast versions of Turing's ideas rather than to the original ones.[12]

Notoriously, neither von Neumann nor Turing himself suggested this connection in 1945 in their original proposals. In a brief note apparently drafted

[a] Newman repeated this claim in an oft-cited oral interview of 1976, but, curiously, in 1955 he wrote a memoir on Turing for the Royal Society in which this point was not mentioned at all.[21]

in 1945 while working on his proposal, Turing implicitly clarified that the original context of the ideas from 1936 could not allow for thinking about a physical calculator, as proposed on the basis of electronic components. He did not mention the issue of instructions stored as data, that might be seen, at least retrospectively, as connecting the current design with the idea of a UTM. Rather, Turing referred only to the issue of accessing the data in reasonable time:

"In 'Computable Numbers' it was assumed that all the stored material was arranged linearly, so in effect the accessibility time was directly proportional to the amount of material stored, being essentially the digit time multiplied by the number of digits stored. This was the essential reason why the arrangement in 'Computable Numbers' could not be taken over as it stood to give a practical form of machine."[7]

But beyond the questionable parallel between a UTM and a stored-program machine, there are more purely historical questions that require clarification. Of particular interest is the actual, direct influence of Turing's paper on von Neumann at the time when the latter wrote his famous "First Draft." Some authors have recently approached this issue and shown (convincingly, in my opinion) that, to the extent von Neumann (or even Turing himself) actually took inspiration from Turing's 1936 paper when engaged in the design of a stored-program computer, these ideas provided at most additional input (arguably not decisive) that was incorporated into a broader, complex array of (mostly engineering and only partly mathematical) considerations; see, for example, Daylight,[12] Haigh,[15] and Haigh et al.[16,b] To what has been said in such works, I add only some specific remarks here. But I think my analysis, by focusing on the earlier part of the story, naturally connects to the views expressed therein and gives them greater credence.

I do not explore the important issue of contemporary developments in different national settings.[3] I limit myself to Turing's work before being recruited to Bletchley Park in September 1939, and I do so by relying on a variety of already published, mostly well-known, primary sources. I argue that the "machines" Turing discussed at the time were purely mathematical constructs. They were not conceived as possible blueprints for building physical calculators. Moreover, I claim the very idea of a modern computer in the sense of either von Neumann's "First Draft" or of Turing's "Proposed Electronic Calculator" was in 1936 not only beyond the scope of Turing's capabilities but also of his concerns. This is true in the obvious (yet crucial) sense that the specific electronic technology that would allow their construction was then beyond Turing's horizon but also true in the less-obvious sense of the question: What should an automatic calculating device be in the first place?

Turing did become involved during this time in the construction of two actual devices, and in both cases the idea of a UTM was of no use, provided no inspiration, and was not even remotely mentioned or hinted at. Neither did Turing suggest that the right approach to building a real computing machine would be along the lines of the UTM and that he would not pursue this direction just for reasons related to technical limitations or to lack of time.

Turing's Computers

Let us start with the 1936 paper itself. Beyond superficial appearance, there is nothing in the original text of "Computable Numbers" that might indicate Turing was referring to, or had in mind, actual physical devices as part of his analysis. The "computers" he referred to in the article were humans who calculate. The aim was to "construct a machine to do the work of this computer," and this is what his famous "machines" were indeed meant to do. Clearly, "construct" was not intended in this text as "physically construct," just as it was not intended when Turing speaks about "constructing" a proof "by the methods of Hilbert and Bernays" or "constructing" a number about which he asks whether or not it is "computable."

The non-physical spirit of Turing's conception of "machines" is highlighted by specific comments that do involve what could be taken on first sight to mean physical components (such as ink or a square) in the (infinite) paper ribbon but which are actually treated as truly abstract entities, when, for instance, he explains the assumption that the number of symbols that may be printed is finite. The reason for this assumption is that otherwise we would have, as Turing said, "symbols that differ from each other to an arbitrarily small extent." A footnote explaining this comment leaves no doubt that, in spite of the wording, Turing is thinking here not as an engineer but purely as a mathematician analyzing the situation with conceptual tools taken from measure theory and topology:

"If we regard a symbol as literally printed on a square we may suppose that the square is $0 \leq x \leq 1$, $0 \leq y \leq 1$. The symbol is defined as a set of points in this square, viz. the set occupied by printer's ink. If these sets are restricted to be measurable, we can define the 'distance' between two symbols as the cost of transforming one symbol into the other if the cost of moving unit area of printer's ink unit distance is unity, and there is an infinite supply of ink at $x = 2$, $y = 0$. With this topology, the symbols form a conditionally compact space."[27]

Some time around May 1936 Turing prepared a two-page French summary of "Computable Numbers." This time he did not mention human computers, explicitly stating that a number may be called "computable" if its decimals can be written by a machine. In describing what he meant by a machine, he explicitly characterized the various configurations of which the machine is "susceptible" as different arrangements of "the levers, the wheels, etc."[c] But even when he used this suggestive wording, it is clear he was speaking figuratively, and that the machines in question were to his mind purely mathematical entities. We can see this from what he writes further:

"A real 'computing machine' should be able to write as many digits as one wishes; "A machine M is called 'malicious' (*méchant*) [what in English he called 'circular'] if there is a number N such that M will never write N digits; and "An application of Cantor's diago-

[b] It should be remarked that the term "stored-program" was introduced in 1949 by IBM engineers in Poughkeepsie, NY.[11]

[c] See http://www.turingarchive.org/browse.php/K/4 In the quotation from Copeland and Sommaruga,[9] the authors refer to this French summary as evidence for their statement about a "three-dimensional machine."

nal argument proves that there exists no machine that, if provided with a description of an arbitrary machine *M*, can decide if *M* is malicious."

It would make little sense to say that by appealing to this abstract argument, used to tackle situations involving infinite sets, Turing intended to say something about an actual physical device.

We can take this analysis one step further by considering Turing's ideas in the context of the well-known contemporary debates of the mid-1930s involving other logicians who came up with their own attempts to provide rigorous mathematical formulations of ideas related to the general notion of "effective computability" or "mechanical procedure." Kurt Gödel, Alonzo Church, Stephen Cole Kleene, Paul Bernays, and Emil Post are among the most prominent names associated with this "period of confluence." Their motivations, the specific problems they were addressing, and the approach they followed were slightly different in each case, and I am unable to delve into them here; see, for example, DeMol[13] and Sieg.[22] For all of them though, the search for mathematically precise concepts, corresponding to what were then informal ideas, only vaguely understood, was crucial.

Turing was the first to introduce into such discourse the word "machine." The more specific term "Turing Machine" was coined by Church in a famous review from 1937. Church, as is well known, had completed at roughly the same time—but following a rather different approach—his own contribution to solving Hilbert's *Entscheidungsproblem*, which was also the focus of Turing's paper. After becoming aware of this, Turing sent it for publication in August 1936, with an appendix proving the equivalence of both approaches and of the ensuing results. In the review, Church described the "Turing Machines" as follows:

"[Turing] proposes as a criterion that an infinite sequence of digits 0 and 1 be 'computable' that it shall be possible to devise a computing machine, occupying a finite space and with working parts of infinite size, which will write down the sequence to any desired number of terms if allowed to run for a sufficiently long time. As a matter of convenience, certain further restrictions are imposed on the character of the machine, but these are of such a nature as obviously to cause no loss of generality—in particular, a human calculator, provided with pencil and paper and explicit instructions, can be regarded as a kind of Turing machine."[4]

Between publication of "Computable Numbers" and his recruitment to Bletchley, Turing was involved in the design and possible construction of two different physical calculators.

Note that Church used the term "computing machine" to mean any calculating machine of finite size, rather than the specific kind of "machines" introduced by Turing. The latter are then characterized by further restrictions, and Turing's human calculator becomes a particular example of the Turing machine. This goes in just the opposite direction from Turing in his original formulation. Moreover, as Hodges[20] wrote, Turing had not referred to machines of finite size as Church did here, and certainly did not define computability in terms of the alleged power of finite machines. Such machines would eventually repeat themselves, and Turing had attempted precisely to show how a machine with finite specifications would not be constrained to do so. The finiteness of Turing's machines concerned only the number of configurations, but the tape, for instance, could not be limited. So we see that Church's account somewhat obscured rather than clarified Turing's powerful, original point of view.

Turing, however, never seems to have explicitly reacted negatively to Church's characterization. Neither did he react to similar remarks by Gödel, who characterized Turing's work in the 1930s as a general analysis of arbitrary machines.[22] It is likely that Turing did not consider such accounts of his ideas as totally unreasonable. But, as Hodges further suggested,[20] it is likely that the participants in this discourse about computability were using the word "machine" in a loose manner and without qualifications to signify "mechanical processes" in general. The machines and the mechanical procedures they referred to were conceived as part of the meta-mathematical attempt to provide the rigorous mathematical characterization of the informal idea of computing rather than suggesting, in any way, some practical guidance on how to build a physical machine.

Yet another contemporary account emphasizes from a different direction the purely mathematical character of Turing's view of his machines, this one by British mathematician Alister Watson. Watson, who, like Turing, was a fellow at King's College and also the person who introduced Turing to Ludwig Wittgenstein in the summer of 1937. What interests us here is Watson's description of Turing's machines in an article he published in 1938:

"Turing's theory of computable numbers is essentially that of mathematical expressions, but he has put it in a rather striking way in terms of machines, which would calculate decimals in accordance with rules that correspond to different mathematical expressions for sequences of this kind. He shows how each such machine can be given a number, different for each machine, and so concludes that the machines and therefore the numbers calculated by them form an enumerable set. Although we can give every machine a number, it is impossible to give a mechanical method by which we can ascertain whether any particular machine is really [circle-free] ..."[31]

Turing's Thesis

Invited to take his Ph.D. under the direction of Church, Turing worked at the Institute for Advanced Study in Princeton from September 1936 to July 1938. His dissertation provides further insight into the relationship between his "machines" and any thoughts he might have had about building an actual physical calculator.

A main innovation of the thesis is the idea of an "oracle" that "cannot be a machine" and which, by definition, involves "some unspecified means of solving number theoretic problems." With the help of the oracle, Turing thus wrote, "We could form a new kind of machine (call them *o*-machines), having as one of its fundamental processes that of solving a given number theoretic problem."

Turing in this context used the term "number theoretic problem" with a precise meaning, namely, problems involving statements of the form "for all ..., there exists ..." He proposed the metamathematical task of establishing the completeness of such problems. The twin prime conjecture and the statement of Fermat's last theorem fall within the scope of these problems. But of special importance for Turing was the fact that the Riemann Hypothesis was also a "number theoretic" problem in his sense. As American logician Solomon Feferman indicated,[14] it is not really clear why Turing concentrated specifically on such statements rather than on arithmetical statements in general. After all, there are other important problems, including the finiteness of the number of solutions of a diophantine equation or the statement of Waring's problem, that do not fit into this definition.

At this stage, Turing was already involved in the design of two actual calculating devices, as I explore in the following sections. And yet, even more than in "Computable Numbers," there is nothing in the way "machines" are referred to in Turing's thesis that may be taken to suggest the idea of building an actual device. Much less does the text suggest that a UTM should be taken as the most appropriate basis for building some kind of "general purpose" or "stored program" calculator. On the other hand, since the "oracle" by its very nature, cannot be a machine in the restricted sense of the Turing machine, it brings to the fore the idea of various possible ways to conceive appropriate models of addressing different mathematical situations. So, in exploring the capabilities of the *o*-machines, Turing actually meant to explore aspects of mathematical proof and of calculation that would not be covered by the machine as defined in 1936.

Tide predicting machine.

IMAGE COURTESY OF NATIONAL OCEANOGRAPHY CENTRE, LIVERPOOL, UK

Turing's Princeton

Turing's encounter with von Neumann at Princeton is one topic that arises repeatedly in texts by scholars who argue for the connection between a UTM and the stored-program computer. A most explicit example of this appears in the following passage from 2011, cited from a text by Jack Copeland and Diane Proudfoot:

"John von Neumann shared Turing's dream of building a universal stored-program computing machine. Von Neumann had learned of the universal Turing machine before the war; he and Turing came to know each other during 1936–1938 when both were at Princeton University."[8]

Indeed, the range of the mutual mathematical interests of these two bright men was very broad. Von Neumann in the 1920s had been a leading figure in the Hilbert circle and Hilbert's collaborator in matters related to the foundations of mathematics. In 1933 he became professor of mathematics at the Institute for Advanced Study. Turing met him in the summer of 1935 in Cambridge when von Neumann lectured there on "almost periodic functions." This was a topic of interest to Turing at the time, and he most certainly attended the course.

Given the later prominence of both Turing and von Neumann in the development of the modern computer, it may seem natural to assume their encounter in Princeton was a period of intense intellectual interchange, particularly around the possibility of building calculating machines. However, a closer look at the evidence tells a completely different story of the encounter and of its relevance to our story. Take for example the following passage from Turing's letter written October 6, 1936, soon after his arrival at Princeton:

"The mathematics department here comes fully up to expectations. There is a great number of the most distinguished mathematicians here. J.v.Neumann, Weyl, Courant, Hardy, Einstein, Lefschetz, as well as hosts of smaller fry. Unfortunately, there are not nearly so many logic people here as last year. Church is here of course, but Gödel, Kleene, Rosser and Bernays ... have left. I don't think I mind very much missing any of these except Gödel."[6]

Turing did not include von Neumann among the "logic people," and with good reason. Right after becoming aware of Gödel's results in 1930, von Neumann deliberately abandoned his previous, very active and important involvement with the foundations of mathematics.[1] The prominent mathematicians listed in Turing's letter showed little interest in the newcomer from Cambridge and in his work on logic. Von Neumann was no exception, nor was Godfrey H. Hardy, who was on visit at the IAS. It is fair to say that Church was Turing's only real interlocutor on logic while Turing was at Princeton.

Both Turing and Church were far from the overly extroverted style of von Neumann, and all evidence indicates there was no personal or friendly relationship with him. We do know Church's few active attempts to make Turing's work better known in Princeton were not particularly successful. Shortly before "Computable Numbers" was published in 1937, Church urged Turing to deliver a talk before the distinguished local mathematical community. Obviously, Turing was thrilled about the opportunity and thought it might bring his work to greater attention. However, it all ended up in disappointment, as we read in one of his letters:

"There was rather bad attendance at the Maths Club for my lecture on Dec. 2. One should have a reputation if one hopes to be listened to."[6]

Turing was also disappointed by the rather limited reaction—besides Church's review essay—aroused by the publication of his paper at the end of 1936. We know that only two persons requested offprints. Even Hermann Weyl, who had been a most prominent member of Hilbert's inner circle and a main figure in the late-1920s debates around the Hilbert program, made not a single remark about the paper. Naturally, Turing was particularly disappointed by Weyl's lack of reaction.[21] And it seems that he did not expect von Neumann to react in any way to his paper. To be sure, besides the letter, von Neumann is not mentioned in any of the letters Turing wrote from Princeton in 1936–1937 to either his mother or his teacher, Philip Hall.

In April 1938, von Neumann approached his younger colleague to offer him a job as assistant, and Turing turned it down. His fellowship at Cambridge had just been renewed, and he was not eager to remain in the U.S. anyway. These may have been the main reasons for Turing's decision. But what about von Neumann's reasons for approaching Turing? Hodges[21] suggested that by this time von Neumann "was aware of 'Computable Numbers,' even if he had not been a year earlier." This is likely, though there is no direct evidence for it. But what is more than evident is that the offer had nothing to do with a direct interest in Turing's work on computability and logic, either as developed in the now famous article of 1936 or as then pursued in his Ph.D. dissertation.[28]

Indeed, back in June 1937, von Neumann had written a letter of recommendation on behalf of Turing for the Procter Fellowship, indicating Turing "had done good work in branches of mathematics in which I am interested, namely: theory of almost periodic functions and theory of continuous groups." Von Neumann, let me emphasize again, had by then completely abandoned his interest in logic, and there is no indication that at the time (and indeed anytime before he became involved in the war effort) he had in any way started to think about computing machines or even about mathematical topics related to massive calculations.[1]

If, as Copeland and Proudfoot[8] emphasized, "von Neumann had learned of the universal Turing machine before the war," there is no indication he devoted special attention to it. While obviously, according to Copeland and Proudfoot, "he and Turing came to know each other... at Princeton," their interaction was rather limited in scope and intensity. There is no indication that the two devoted any time to discussing Turing's ideas on the topic and much less that they shared (or discussed) during these years anything like a "dream of building a universal stored-program computing machine."

Turing's Analog Machine

Between publication of "Computable Numbers" and his recruitment to Bletchley, Turing was involved in the design and possible construction of two different physical calculators. The first, in the Fall of 1937, was essentially an electric multiplier. This was for Turing an early and rather rudimentary (though no doubt original) foray into machine-based cryptanalysis. On the mathematical side, it appealed to a simple but theretofore not very well noticed parallel between binary arithmetic and Boolean algebra. On the physical side, it brought to bear the possibility of using, as the basis for a computing device, the kind of electromagnetic relays that had already been in use for approximately 100 years in the context of telegraphy.

In his account of this interesting episode, Hodges[19] explained it as an attempt, on the side of Turing, to build a physical embodiment of a specific "Turing Machine" meant to deal with a specific mathematical problem, with the network of relay-operated switches acting as material counterparts of the "configurations":

"The idea would be that when a number was presented to the machine, presumably by setting up currents at a series of input terminals, the relays would click open and closed, currents would pass through, and emerge at output terminals, thus in effect 'writing' the enciphered number."[21]

We have no evidence that Turing himself would have described in these terms what he was doing here or that, in his view, with the relay multiplier, "'Turing machines' were coming to life," as Hodges further remarked. But one way or another, as Hodges himself clearly stated elsewhere,[19] "This offbeat amateur engineering was the closest Turing came to developing his ideas of general computation in a practical direction." This episode underscores, in my view, the unlikeliness of seeing Turing's 1936 UTM as a blueprint for a counterpart physical device that would be general-purpose, digital, and, more important, stored-program. An engineering project involving such ideas was simply well beyond the horizon of what Turing had in mind at this point.

But all such intent becomes even more evident when we take a close look at a second calculating device Turing designed and started to build in 1939. This was one specifically conceived for calculating approximate values of the Riemann zeta-function on its critical line. Hodges's Turing website "Alan Turing: the Enigma" (http://www.turing.org.uk/index.html), the foremost repository of scholarly information for anyone interested in Turing's life and work, displays the application submitted by Turing to the Royal Society for a grant support for building the device, as well as a blueprint with some details about its technical design.[d] This is not the place for details about the mathematical significance of the Riemann Hypothesis, or RH, of Turing's overall involvement with this problem, or of his specific contribution to it; such information can be found in Booker[2] and in Hejhala and Odlyzko.[17] But there are some specific points that are highly relevant to our concerns here and proceed to discuss them now.

Turing's interest in RH was sparked shortly after matriculating at King's College in 1931. There he attended a course by Albert E. Ingham, who in 1932 published an important text on the distribution of prime numbers. At King's, Turing also befriended Stanley Skewes, who in 1933 made a remarkable contribution to research on RH. Briefly, Skewes calculated an upper bound for the smallest value of x for which $\pi(x) > \mathrm{Li}(x)$. Here $\pi(x)$ represents the number of primes that are smaller than a given integer x, while $\mathrm{Li}(x)$ is the value of the integral $\int_2^x \frac{\mathrm{d}x}{\ln x}$. It is well known that RH concerns the question of the nontrivial zeros of the Riemann zeta-function $\zeta(s)$ and its relation to the estimation of the value of the difference between the two functions. John E. Littlewood had proved, in 1912, contrary to the common belief at the time, that the difference $\pi(x) - \mathrm{Li}(x)$ changes sign infinitely many times, both if RH is true and if RH is false. Skewes's proof involved two different values of the upper bound for the respective cases, and both were amazingly high. For instance, if RH is true, then he proved that the smallest number x where this change of sign happens is smaller than $10^{10^{10^{34}}}$.

While at Princeton, and during his visit to Cambridge in the Summer of 1937, Turing actively pursued research related to RH. Among other things, he improved the value of Skewes's bound, and, more important, improved the existing methods for calculating zeros of $\zeta(s)$. Shortly before that, Edward Ch. Titchmarsch had been involved with such calculations at Oxford and was able to establish that the first 1,041 nontrivial zeros of $\zeta(s)$ all satisfy RH.[24,25] Turing's plan from 1939 for building a calculating device is directly connected to Titchmarsch's work.

Titchmarsch's calculations were based on approximation formulas that required massive iterations of addition and multiplication, as well as the use of cosine tables. For "planning and supervising the calculations, which were carried out with Brunsviga, National, and Hollerith machines," Titchmarsch explicitly thanked J.L. Comrie,[25] who had, since 1929, been "Secretary of the British Mathematical Tables Committee" and the driving force behind a great amount of important projects of scientific table making conducted in the U.K. at the time.[10]

Astronomical tables were of particular importance among such projects, and their preparation involved repetitive summations of circular functions involving different frequencies for plotting the positions of planets. Comrie's recent introduction of Hollerith punch-card techniques to scientific table making[5] signified a remarkable innovation in the field. As it happened, the calculations required in Titchmarsch's approach to calculating zeros of $\zeta(s)$ were quite similar to them. Comrie was clearly the right person to provide the necessary technical assistance.

Comrie did not typically perform any calculations himself. Rather, he had teams of "human computers" (mainly women) to do that work. Each computer received data to be worked out, and, with the help of "Brunsviga, National, and Hollerith machines," would perform a specific task, clearly defined in advance according to the kind of input presented to her. She would then deliver the result to another person in the team who also carried out a respective task, well defined in advance, again depending on the kind of input at stake. Note while Turing's "machines" of 1936 provided a mathematical model meant to analyze the nature and scope of the calculations that individual human computers could and did effectively perform, those machines can also be taken to describe adequately (and perhaps even better) the model of what coordinated teams of human computers were actually doing in Turing's time when addressing heavy computational tasks that went beyond what an individual could achieve.

When Turing in 1939 undertook to perform relatively massive calculations in one of his main mathematical topics of interest other than logic—the Riemann Hypothesis—he was aware that precisely at that time, one of Comrie's teams had recently been involved in the same task, with the aid of mechanical calculators. The conceptual similarity between Turing's model and Comrie's activities could not have escaped Turing's notice. And yet, for his own calculations, he did not choose to follow or improve the direction previously pursued by Titchmarsch. Neither did he take a step further toward anything like a UTM. Rather, he went into the totally different direction of designing an analog machine, specifically conceived for supporting his work on what for him was such an important mathematical task.

Turing's analog design relied on a machine built some years earlier for tide prediction at Liverpool, England. It performed trigonometrical summations based on a combination of pulley wheels, each representing one of the gravitational effects that give rise to tidal phenomena. A thin nickel tape connected the various pulley wheels and "summed up," as it were, their separate movements. The tidal highs and lows were thereby registered on a chart located at the bottom of the machine.

Turing thought similar principles would be useful for constructing an apparatus for calculating trigonometric sums that were at the heart of his method for the zeros of $\zeta(s)$. In some cases, he thought, the apparatus would not be accurate enough and it would then be necessary to work out the calculations manually. But he believed such cases would be extremely rare. He specifically stressed that the apparatus would be so closely analog to the simulated mathematical situation to the extent that he

[d] See http://www.turing.org.uk/sources/zetamachine.html

could not think "of any application that would not be connected with" $\zeta(s)$. The project, at any rate, was never completed because of the outbreak of war, and none of its parts has survived.

I find it quite interesting that when Hodges describes the project on his website, he mentions "a special machine to calculate approximate values for the Riemann zeta-function on its critical line." This is not the only place where the term "special machine," which was not used by Turing, is used in this context, as in, for example, Booker[2] and Hejhala and Odlyzko.[17] I take it to be a revealing, subtly misleading description of what Turing had in mind in 1939, particularly because the project is often mentioned in conjunction with Turing's later efforts of 1950 to attack the same problem using a "general purpose" electronic computer, the Manchester University Mark I.[29]

My point is that in 1939 Turing approached the actual construction of an apparatus the way he did, not because—for lack of time or resources—he was compelled to make do with it. This was not a "special" limited or rudimentary version of what for him would be the real thing, namely, a general-purpose, stored-program, digital and electronic computer (presumably being a physical embodiment of a UTM). Rather, it is that a putative physical version of a UTM was not within the horizon of possible or convenient approaches to be followed. To the contrary, what the evidence shows is that, at the time, Turing considered the analog approach to be the most intrinsically appropriate for the task at hand.

In fact, the tremendous success of modern digital computers has negatively affected the way the history of analog computers in general has been told. The case of Turing's 1939 project is just one example of such retrospective misreading, though one seldom mentioned in this context. Analog computers were not only a natural choice in many situations before the war but even after the emergence of digital computing in the post-war period were not immediately displaced.[23] A most remarkable example of this is precisely the Liverpool tide-predicting machine, which remained in use until the 1960s, before being superseded by electronic computers.

By its very nature, Turing's oracle could not be a standard Turing machine.

Intuition and Turing's Machines

Turing left Cambridge in September 1939 for Bletchley Park, and would over the next few years devote all of his intellectual energies to war-related projects while temporarily setting aside mathematical research in his other fields of interest. By 1945 he would have added important skills, as well as familiarity with electronic valves and with machines of various kinds, to the already impressive arsenal of knowledge he had brought with him at the time of his recruitment. His activities after 1945, including, of course, all of his involvement with electronic computers, were deeply influenced by his years at Bletchley.

But over the first months following his recruitment, Turing and Newman continued to correspond and discuss issues connected to their pre-war activity. This correspondence bears witness to the way the two went on to speak about "Turing machines" at a time when Turing had already gained actual experience, not only with the two calculating devices he had been involved with, but also with what he had now started to learn at Bletchley. Remarkably, they stuck to the language of "machines," not in the sense of physical devices but rather as the relevant, purely mathematical idea on the basis of which they would discuss issues related to the foundations of arithmetic.

In a particularly interesting passage, under the heading "Ingenuity and Intuition," Turing replied to a previous letter from Newman in which Newman seems to have commented on matters related to the Hilbert program:

"I think you take a much more radically Hilbertian attitude about mathematics than I do. You say 'If all this whole formal outfit is not about finding proofs which can be checked on a machine it's difficult to know what it is about.' [D]o you have in mind that there is (or should be or could be, but has not been actually described anywhere) some fixed machine ... and that the formal outfit is, as it were about this machine. If you take this attitude ... there is little more to be said: we simply have to get used to the technique of this machine and resign ourselves to the fact that there are some problems to which we can never get the answer ... If you think of various machines I don't see your difficulty. One imagines different machines allowing different sets of proofs, and by choosing a suitable machine one can approximate 'truth' by 'provability' better than with a less suitable machine, and can in a sense approximate it as well as you please. The choice of a machine involves intuition,... or as [an] alternative one may go straight for the proof and this again requires intuition."[6]

The fact that still, in 1940, when the classical debates on the foundations of arithmetic had almost totally faded away, Newman and Turing continued their exchanges on such matters, is worthy of attention in itself. But no less interesting is the subtle twist Turing introduced into this discussion when he mentioned the possibility of having various kinds of machines according to different kinds of intuitions that are relevant to different mathematical situations. The UTM was not for Turing "universal" in this important sense.

It seems that now—in those few opportunities when he could think about the foundations of mathematics and about questions of "truth" or "provability"—Turing also incorporated new directions (such as he had explored in his Ph.D. dissertation). This included, no doubt, the oracle, but also, so it seems, alternatives to the basic "machine" he had defined in 1936.

Conclusion

I conclude with a final, somewhat conjectural suggestion. By its very nature, Turing's oracle could not be a standard Turing machine. "Solving a given number theoretic problem" is one of "its fundamental processes." And the Riemann Hypothesis is one such problem. Now, it seems to me, Turing's construction of his analog machine and the variety of machines he mentioned in his response to Newman shed interesting light, retrospectively, on that passing, somewhat unclear comment Turing advanced in his thesis. From the letter we learn that each mathematical situation calls for the choice of a suitable machine and these choices rely on the right intuition to do so in each case. The UTM had been a highly successful, specific choice for dealing with the Entscheidungsproblem, but that would not mean—even in principle—it would provide a model for a physical universal machine, suitable for all mathematical tasks. If we may somehow think of these mathematical models as suggesting blueprints for designing physical devices, then an analog machine (such as planned by Turing in 1939) would come much closer than anything built along the lines of a UTM to embodying specific, "fundamental processes" associated with a particular number theoretic problem, in the sense suggested in his Ph.D. thesis.

When Turing in 1950 returned to the task of calculating zeros of $\zeta(s)$ the sea changes that had revolutionized the world of automatic computation had rendered all those pre-war considerations obsolete. The natural approach to follow for Turing was now to write a specific program to run in a stored-program, general-purpose machine. But the Mark I, like all other similar machines at the time, was not only stored-program. It was also electronic, large-scale, high-speed, general purpose, and digital. In 1939, all these crucial components of the machines that started to be built in the late 1940s were far beyond the horizon.

Acknowledgments

This article is a somewhat belated elaboration of a talk I gave at the "Turing in Context II" conference in Brussels, October 10–12, 2012 (http://www.dijkstrascry.com/Corry). I want to thank Edgar Daylight for transcribing the talk and for his continued encouragement to publish my ideas on this topic. Thanks, too, to three anonymous referees for valuable comments.

References

1. Aspray, W. *John von Neumann and the Origins of Modern Computing.* MIT Press, Cambridge, MA, 1990.
2. Booker, A.R. Turing and the primes. In *The Once and Future Turing: Computing the Worldy*, S.B. Cooper and A. Hodges, Eds. Cambridge University Press, Cambridge, U.K., 2016, 34–50.
3. Bruderer, H. *Meilensteine der Rechentechnik. Zur Geschichte der Mathematik und der Informatik.* De Gruyter Oldenbourg, Berlin/Boston, 2015.
4. Church, A. Review: A.M. Turing, on computable numbers, with an application to the *Entscheidungsproblem. Journal of Symbolic Logic 2*, 1 (1937), 42–43.
5. Comrie, L.J. The application of the Hollerith tabulating machine to Brown's tables of the moon. *Monthly Notices of the Royal Astronomical Society 92*, 7 (1932), 694–707.
6. Copeland, B.J., Ed. *The Essential Turing.* Clarendon Press, Oxford, 2004.
7. Copeland, B.J., Ed. *Alan Turing's Automatic Computing Engine. The Master Codebreaker's Struggle to Build the Modern Computer.* Oxford University Press, Oxford, U.K., 2005.
8. Copeland, B.J. and Proudfoot, D. Alan Turing: Father of the modern computer. *Rutherford Journal 4* (2011–2012).
9. Copeland, B.J. and Sommaruga, G. The stored-program universal computer: Did Zuse anticipate Turing and von Neumann? In *Turing's Revolution. The Impact of His Ideas about Computability*, G. Sommaruga and T. Strahm, Eds. Springer International Publishing, Cham, Switzerland, 2015, 43–101.
10. Croarken, M. Table making by committee: British table makers 1871–1965. In *The History of Mathematical Tables: From Sumer to Spreadsheets*, R.F.M. Campbell-Kelly, M. Croarken, and E. Robson, Eds. Oxford University Press, Oxford, U.K., 2003, 235–267.
11. Daylight, E.G. A Turing tale. *Commun. ACM, 57*, 10 (Oct. 2014), 36–38.
12. Daylight, E.G. Towards a historical notion of 'Turing: The father of computer science.' *History and Philosophy of Logic 36*, 3 (Sept. 2015), 205–228.
13. DeMol, L. Closing the circle: An analysis of Emil Post's early work. *The Bulletin of Symbolic Logic 12*, 2 (June 2006), 267–289.
14. Feferman, S. Turing's thesis. *Notices of the American Mathematical Society 53*, 10 (Nov. 2006), 2–8.
15. Haigh, T. Actually, Turing did not invent the computer. *Commun. ACM 57*, 1 (Jan. 2014), 36–41.
16. Haigh, T., Priestley, M., and Rope, C. Reconsidering the stored-program concept. *IEEE Annals of the History of Computing 36*, 1 (Jan.-Mar. 2014), 4–17.
17. Hejhala, D.A. and Odlyzko, A.M. Alan Turing and the Riemann zeta function. In *Alan Turing: His Work and Impact*, S.B. Cooper and J. van Leeuwen, Eds. Elsevier Science, Waltham, MA, 2013, 265–279.
18. Hodges, A. *The Alan Turing Internet Scrapbook. Who Invented the Computer?*; http://www.turing.org.uk/scrapbook/computer.html
19. Hodges, A. Alan Turing: The logical and physical basis of computing. In *Proceedings of the 2004 International Conference on Alan Mathison Turing: A Celebration of his Life and Achievements* (Swinton, U.K.). British Computer Society, London, U.K., 2004.
20. Hodges, A. Did Turing have a thesis about machines? In *Church's Thesis After 70 Years*, J.W.A. Olszewski and R. Janusz, Eds. Ontos Verlag, Frankfurt am Main, Germany, 2006, 242–252.
21. Hodges, A. *Alan Turing: The Enigma. The Book that Inspired the Film 'The Imitation Game.'* Princeton University Press, Princeton, NJ, 2014.
22. Sieg, W. Gödel on computability. *Philosophia Mathematica 14*, 2 (Jan. 2006), 189–207.
23. Small, J.S. *The Analogue Alternative: The Electric Analogue Computer in Britain and the USA, 1930–1975.* Routledge, London, U.K., 2001.
24. Titchmarsh, E.C. The zeros of the Riemann zeta-function. *Proceedings of the Royal Society of London. Series A, Mathematical, Physical and Engineering Sciences 151*, 873 (Sept. 1935), 234–255.
25. Titchmarsh, E.C. The zeros of the Riemann zeta-function. *Proceedings of the Royal Society of London. Series A, Mathematical, Physical and Engineering Sciences 157*, 891 (Nov. 1936), 261–263.
26. Turing, A.M. *Proposed Electronic Calculator* (report submitted to the Executive Committee of the National Physical Laboratory), 1945; republished in Copeland, B.J, Ed.,[7] 369–454.
27. Turing, A.M. On computable numbers, with an application to the *Entscheidungsproblem. Proceedings of the London Mathematical Society 42*, 1 (Nov. 1937), 230–265.
28. Turing, A.M. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society 45*, 1 (Jan. 1939), 161–228.
29. Turing, A.M. Some calculations of the Riemann zeta-function. *Proceedings of the London Mathematical Society 3*, 3 (Jan. 1953), 99–117.
30. von Neumann, J. *First Draft of a Report on the EDVAC. Technical Report.* University of Pennsylvania, Philadelphia, PA, 1945.
31. Watson A.G.D. Mathematics and its foundations. *Mind 47*, 188 (Oct. 1938), 440–451.

Leo Corry (corry@post.tau.ac.il) is the Bert and Barbara Cohn Professor of History and Philosophy of Science and Dean of the Lester and Sally Entin Faculty of Humanities at Tel-Aviv University.

Watch the author discuss his work in this exclusive *Communications* video. https://cacm.acm.org/videos/turings-pre-war-analog-computers

DOI:10.1145/3015456

While it may not be possible to build a data brain identical to a human, data science can still aspire to imaginative machine thinking.

BY LONGBING CAO

Data Science: Challenges and Directions

WHILE DATA SCIENCE has emerged as an ambitious new scientific field, related debates and discussions have sought to address why science in general needs data science and what even makes data science a science. However, few such discussions concern the intrinsic complexities and intelligence in data science problems and the gaps in and opportunities for data science research. Following a comprehensive literature review,[5,6,10–12,15,18] I offer a number of observations concerning big data and the data science debate. For example, discussion has covered not only data-related disciplines and domains like statistics, computing, and informatics but traditionally less data-related fields and areas like social science and business management as well. Data science has thus emerged as a new inter- and cross-disciplinary field. Although many publications are available, most (likely over 95%) concern existing concepts and topics in statistics, data mining, machine learning, and broad data analytics. This limited view demonstrates how data science has emerged from existing core disciplines, particularly statistics, computing, and informatics. The abuse, misuse, and overuse of the term "data science" is ubiquitous, contributing to the hype, and myths and pitfalls are common.[4] While specific challenges have been covered,[13,16] few scholars have addressed the low-level complexities and problematic nature of data science or contributed deep insight about the intrinsic challenges, directions, and opportunities of data science as an emerging field.

Data science promises new opportunities for scientific research, addressing, say, "What can I do now but could not do before, as when processing large-scale data?"; "What did I do before that does not work now, as in methods that view

» key insights

- Data science problems require systematic thinking, methodologies, and approaches to help spur development of machine intelligence.
- The conceptual landscape of data science assists data scientists trying to understand, represent, and synthesize the complexities and intelligence in related problems.
- Data scientists aim to invent data- and intelligence-driven machines to represent, learn, simulate, reinforce, and transfer human-like intuition, imagination, curiosity, and creative thinking through human-data interaction and cooperation.

data objects as independent and identically distributed variables (IID)?"; "What problems not solved well previously are becoming even more complex, as when quantifying complex behavioral data?''; and "What could I not do better before, as in deep analytics and learning?"

As data science focuses on a systematic understanding of complex data and related business problems,[5,6] I take the view here that data science problems are complex systems[3,19] and data science aims to translate data into insight and intelligence for decision making. Accordingly, I focus on the complexities and intelligence hidden in complex data science problems, along with the research issues and methodologies needed to develop data science from a complex-system perspective.

What It Is

The concept of data science was originally proposed within the statistics and mathematics community[23,24] where it essentially concerned data analysis. Data science today[17] goes beyond specific areas like data mining and machine learning or whether it is the next generation of statistics.[9,11,12] But what is it?

Definition. Data science is a new trans-disciplinary field that builds on and synthesizes a number of relevant disciplines and bodies of knowledge, including statistics, informatics, computing, communication, management, and sociology, to study data following "data science thinking"[6] (see Figure 1). Consider this discipline-based data science formula

$$\text{data science} = \{\text{statistics} \cap \text{informatics} \cap \text{computing} \cap \text{communication} \cap \text{sociology} \cap \text{management} \mid \text{data} \cap \text{domain} \cap \text{thinking}\}$$

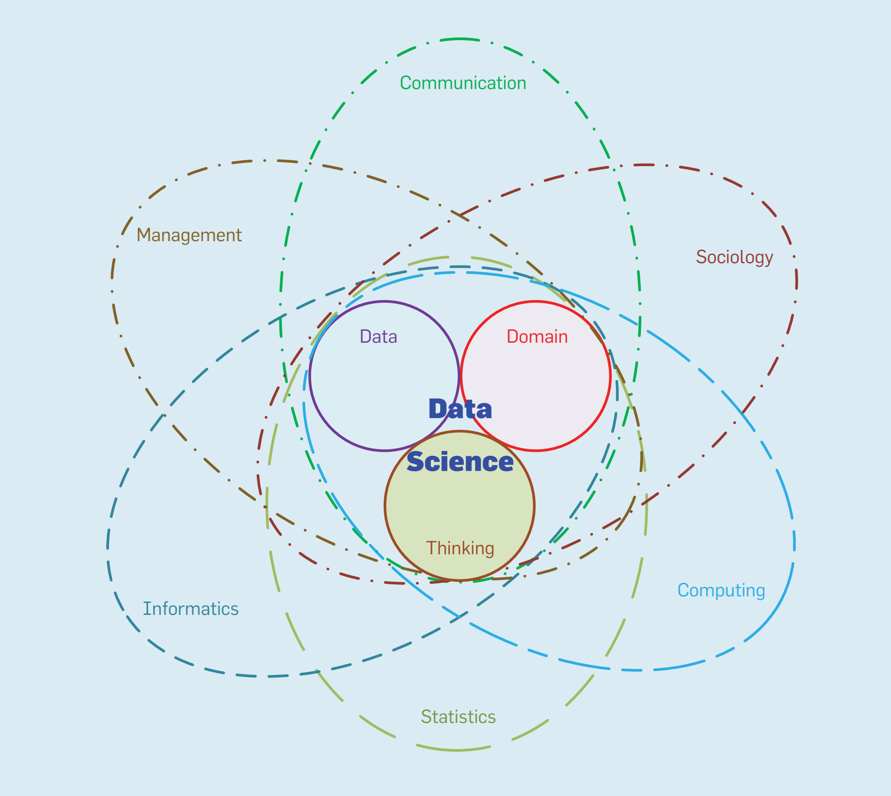
where "|" means "conditional on."

X-Complexities in Data Science

A core objective of data science is exploration of the complexities[19] inherently trapped in data, business, and problem-solving systems.[3] Here, complexity refers to sophisticated characteristics in data science systems. I treat data science problems as complex systems involving comprehensive system complexities, or X-complexities, in terms of data (characteristics), behavior, domain, social factors, environment (context), learning (process and system), and deliverables.

Data complexity is reflected in terms of sophisticated data circumstances and characteristics, including large scale, high dimensionality, extreme imbalance, online and real-time interaction and processing, cross-media applications, mixed sources, strong dynamics, high frequency, uncertainty, noise mixed with data, unclear structures, unclear hierarchy, heterogeneous or unclear distribution, strong sparsity, and unclear availability of specific sometimes critical data. An important issue for data scientists involves the complex relations hidden in data that are critical to understanding the hidden forces in data. Complex relations could consist of comprehensive couplings[2] that may not be describable through existing association, correlation, dependence, and causality theories and systems. Such couplings may include explicit and implicit, structural and nonstructural, semantic and syntactic, hierarchical and vertical, local and global, traditional and nontraditional relations, and evolution and effect.

Figure 1. Transdisciplinary data science.



Data complexities inspire new perspectives that could not have been done or done better before. For example, traditional large surveys of sensor data, including statisticians' questions and survey participants, have been shown to be less effective, as seen in related complications (such as wrongly targeted participants, low overall response rate, and questions unanswered). However, data-driven discovery can help determine who is to be surveyed, what questions need to be answered, the actionable survey operation model, and how cost-effective the survey would be.

Behavior complexity refers to the challenges involved in understanding what actually takes place in business activities by connecting to the semantics and processes and behavioral subjects and objects in the physical world often ignored or simplified in the data world generated by physical-activity-to-data conversion in data-acquisition and -management systems. Behavior complexities are embodied in coupled individual and group behaviors, behavior networking, collective behaviors, behavior divergence and convergence, "nonoccurring"[8] behaviors, behavior-network evolution, group-behavior reasoning, recovery of what actually happened, happens, or will happen in the physical world from the highly deformed information collected in the purely data world, insights, impact, utility, and effect of behaviors, and the emergence and management of behavior intelligence. However, limited systematic research outcomes are available for comprehensively quantifying, representing, analyzing, reasoning about, and managing complex behaviors.

Data scientists increasingly recognize domain complexity[7] as a critical aspect of data science for discovering intrinsic data characteristics, value, and actionable insight. Domain complexities are reflected in a problem domain as domain factors, domain processes, norms, policies, qualitative-versus-quantitative domain knowledge, expert knowledge, hypotheses, meta-knowledge, involvement of and interaction with domain experts and professionals, multiple and cross-domain interactions, experience acquisition, human-machine synthesis, and roles and leadership in the domain. However, existing data analytics focuses mainly on domain knowledge.

Social complexity is embedded in business activity and its related data and is a key part of data and business understanding. It may be embodied in such aspects of business problems as social networking, community emergence, social dynamics, impact evolution, social conventions, social contexts, social cognition, social intelligence, social media, group formation and evolution, group interaction and collaboration, economic and cultural factors, social norms, emotion, sentiment and opinion influence processes, and social issues, including security, privacy, trust, risk, and accountability in social contexts. Promising interdisciplinary opportunities emerge when social science meets data science.

Environment complexity is another important factor in understanding complex data and business problems, as reflected in environmental (contextual) factors, contexts of problems and data, context dynamics, adaptive engagement of contexts, complex contextual interactions between the business environment and data systems, significant changes in business environment and their effect on data systems, and variations and uncertainty in interactions between business data and the business environment. Such aspects of the system environment have concerned open complex systems[20] but not yet data science. If ignored, a model suitable for one domain might produce misleading outcomes in another, as is often seen in recommender systems.

Learning (process and system) complexity must be addressed to achieve the goal of data analytics. Challenges in analyzing data include developing methodologies, common task frameworks, and learning paradigms to handle data, domain, behavioral, social, and environmental complexity. Data scientists must be able to learn from heterogeneous sources and inputs, parallel and distributed inputs, and their infinite dynamics in real time; support on-the-fly active and adaptive learning of large data volumes in computational resource-poor environments (such as embedded sensors), as well as multisource learning, while considering the relations and interactions between sensors; enable combined learning across multiple learning objectives, sources, feature sets, analytical methods, frameworks, and outcomes; learn non-IID data-mixing coupling relationships with heterogeneity;[2] and ensure transparency and certainty of learning models and outcomes.

Other requirements for managing and exploiting data include appropriate design of experiments and mechanisms. Inappropriate learning could result in misleading or harmful outcomes, as in a classifier that works for balanced data but could mistakenly classify biased and sparse cases in anomaly detection.

The complexity of a deliverable data product, or "deliverable complexity" becomes an obstruction when actionable insight[7] is the focus of a data science application. Such complexity necessitates identification and evaluation of the outcomes that satisfy technical significance and have high business value from both an objective and a subjective perspective. The related challenges for data scientists also involve designing the appropriate evaluation, presentation, visualization, refinement, and prescription of learning outcomes and deliverables to satisfy diverse business needs, stakeholders, and decision support. In general, data deliverables to business users must be easy to understand and interpretable by nonprofessionals, revealing insights that directly inform and enable decision making and possibly having a transformative effect on business processes and problem solving.

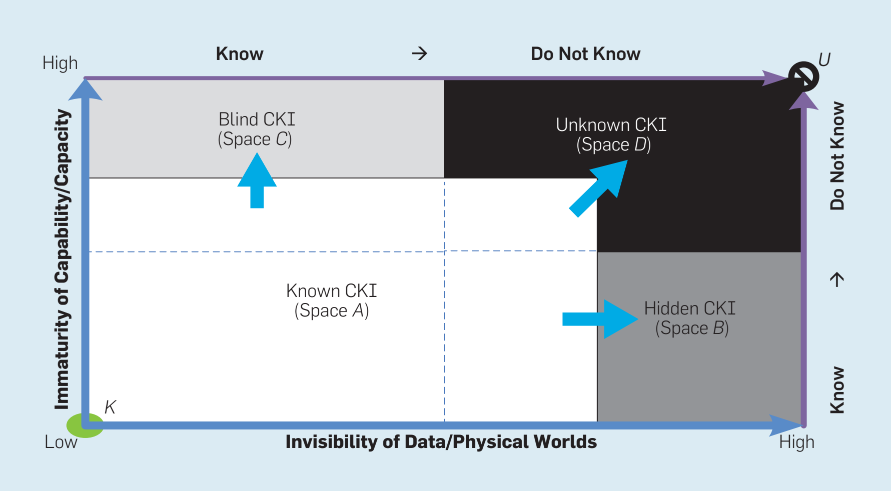
X-Intelligence in Data Science

Data science is a type of "intelligence science" that aims to transform data into knowledge, intelligence, and wisdom.[21] In this transformation, comprehensive intelligence,[3] or "X-intelligence," is often used to address a complex data science problem, referring to comprehensive and valuable information. X-intelligence can help inform the deeper, more structured and organized comprehension, representation, and problem solving in the underlying complexities and challenges.

Data intelligence highlights the most valuable information and narratives in the formation and solution of business problems or value in the corresponding data. Intelligence hidden in data is discovered by data science through its ability to understand data characteristics and complexities. Apart from the usual focus on complexities in data structures, distribution, quantity, speed, and quality, the focus in data science is on the intelligence hidden in the unknown "Space *D*" in Figure 2. For example, in addition to existing protocols for cancer treatment, determining what new and existing treatments fail on which patients might be informed by analyzing healthcare data and diversified external data relevant to cancer patients. The level of data intelligence depends on how much and to what extent a data scientist is able to deeply understand and represent data characteristics and complexities.

Data scientists discover behavior intelligence by looking into the activities, processes, dynamics, and impact of individual and group actors, or the behavior and business quantifiers, owners, and users in the physical world. Such discovery requires they be able to bridge the gap between the data world and the physical world by connecting what happened and what will happen in the problem and discovering behavior insights through behavior informatics.[1] For example, in monitoring online shopping websites, regulators must be able to recognize whether ratings and comments are made by robots, rather than humans;

Figure 2. Known-to-unknown discovery in data science.



likewise, in social media, detecting algorithms, or robot-generated comments, in billions of daily transactions is itself a computational challenge. Constructing sequential behavior vector spaces and modeling interactions with other accounts in a given time period and then differentiating abnormal behaviors may be useful for understanding the difference between proactive and subjective human activity and the reactive and patternable behaviors of software robots.

Domain intelligence emerges from relevant domain factors, knowledge, meta-knowledge, and other domain-specific resources associated with a problem and its target data. Qualitative and quantitative domain intelligence can help inform and enable a data scientist's deep understanding of domain complexities and their roles in discovering unknown knowledge and actionable insight. For example, to learn high-frequency trading strategies for use with stock data, a strategy modeler must include the "order book" and microstructure of the related "limit market."

Human intelligence plays a central role in complex data science systems through explicit, or direct, involvement of human intuition, imagination, empirical knowledge, belief, intention, expectation, runtime supervision, evaluation, and expertise. It also concerns the implicit, or indirect, involvement of human intelligence in the form of imaginative thinking, emotional intelligence, inspiration, brainstorming, reasoning inputs, and embodied cognition, as in convergent thinking through interaction with fellow humans. For example, as "data-science thinking"[6] is crucial for addressing complex data problems, data scientists must be able to apply subjective factors, qualitative reasoning, and critical imagination.

Network intelligence emerges from both Web intelligence and broad-based networking and connected activities and resources, especially through the Internet of Things, social media, and mobile services. Information and facilities from the networks involved in target business problems can contribute useful information for complex data-science problem solving; a relevant example is crowdsourcing-based open source system development and algorithm design.

Organizational intelligence emerges from the proper understanding, involvement, and modeling of organizational goals, actors, and roles, as well as structures, behaviors, evolution and dynamics, governance, regulation, convention, process, and workflow in data science systems. For example, the cost effectiveness of enterprise analytics and functioning of data science teams rely on organizational intelligence.

Social intelligence emerges from the social complexities discussed earlier. Human social intelligence is embedded in social interactions, group goals and intentions, social cognition, emotional intelligence, consensus construction, and group decision making. Social intelligence is also associated with social-network intelligence and collective interactions among social systems, as well as the business rules, law, trust, and reputation for governing social intelligence. Typical artificial social systems include social networks and social media in which data-driven social complexities are understood through social-influence modeling, latent relation modeling, and community formation and evolution in online societies.

Environmental intelligence is also hidden in data science problems, as specified in terms of the underlying domain and related organizational, social, human, and network intelligence. Data science systems are open, with interactions between the world of transformed data and the physical world functioning as the overall data environment. Examples include context-aware analytics involving contextual factors and evolving interactions and changes between data and context, as in infinite-dynamic-relation modeling in social networks.

Known-to-Unknown Transformation

Complex data science problem-solving journeys taken by data scientists represent a cognitive progression from understanding known to unknown complexities in order to transform data into knowledge, intelligence, and insight for decision taking by inventing and applying respective data-intelligence discovery capabilities. In this context, knowledge represents processed information in terms of information mixture, procedural action, or propositional rules; resulting insight refers to the deep understanding of intrinsic complexities and mechanisms in data and its corresponding physical world.

Figure 2 outlines data science progression aiming to reduce the immaturity of capabilities and capacity (y-axis) to better understand the hidden complexities, knowledge, and intelligence (CKI) in data/physical worlds (x axis) from the 100% known state K to the 100% unknown state U. Based on data/physical world visibility and capability/capacity maturity, data science can be categorized into four data challenges:

"Space *A*" represents the known space; that is, "I (my mature capability/capacity) know what I know (about the visible world)." This is like the ability of sighted people to recognize an elephant by seeing the whole animal, whereas non-sighted people might be able to identify only part of the animal through touch. Knowledge concerning visible data is known to people with mature capability/capacity; that is, their capability/capacity maturity is sufficient to understand data/physical-world invisibility. This insight corresponds to well-understood areas in data science. Examples include profiling and descriptive analysis that applies existing models to data deemed by data analysts to follow certain assumptions.

"Space *B*" represents the hidden space; that is, "I know what I do not know (about the unseen world)." For some people or disciplines, even though certain aspects of their capability/capacity is mature, CKI is hidden to (so cannot be addressed by) current data science capability/capacity, thus requiring more-advanced capability/capacity. Examples include existing IID models (such as k-means and the k-nearest neighbors algorithm) that cannot handle non-IID data.

"Space *C*" represents the blind space; that is, "I (my immature capability) do not know what I know (about the world)." Although CKI is visible to some people or disciplines, their capability/capacity is also mature, but CKI and capability/capacity do not match well; immaturity thus renders them blind to the world. An example might be when even established social scientists try to address a data science problem.

"Space *D*" represents the unknown; that is, "I do not know what I do not know, so CKI in the hidden world is unknown due to immature capability." This is the area today on which data science focuses its future research and discovery. Along with increased invisibility, the lack of capability maturity also increases. In the world of fast-evolving big data, CKI invisibility increases, resulting in an ever-larger unknown space.

The current stage of data science capability and maturity, or "We do not know what we do not know," can be explained in terms of unknown perspectives and scenarios. As outlined in Figure 3, the unknown world presents "unknownness" in terms of certain definable categories, including problems and complexities; hierarchy, structures, distributions, relations, and heterogeneities; capabilities, opportunities, and gaps; and solutions.

Data Science Directions

Here, I consider the applied data science conceptual landscape, followed by two significant aspirational goals: non-IID data learning and human-like intelligence.

Data science landscape. The X-complexity and X-intelligence in complex data science systems and widening gap between world invisibility and capability/capacity immaturity yield new research challenges that motivate development of data science as a discipline. Figure 4 outlines the conceptual

Figure 3. Hidden world in data science.

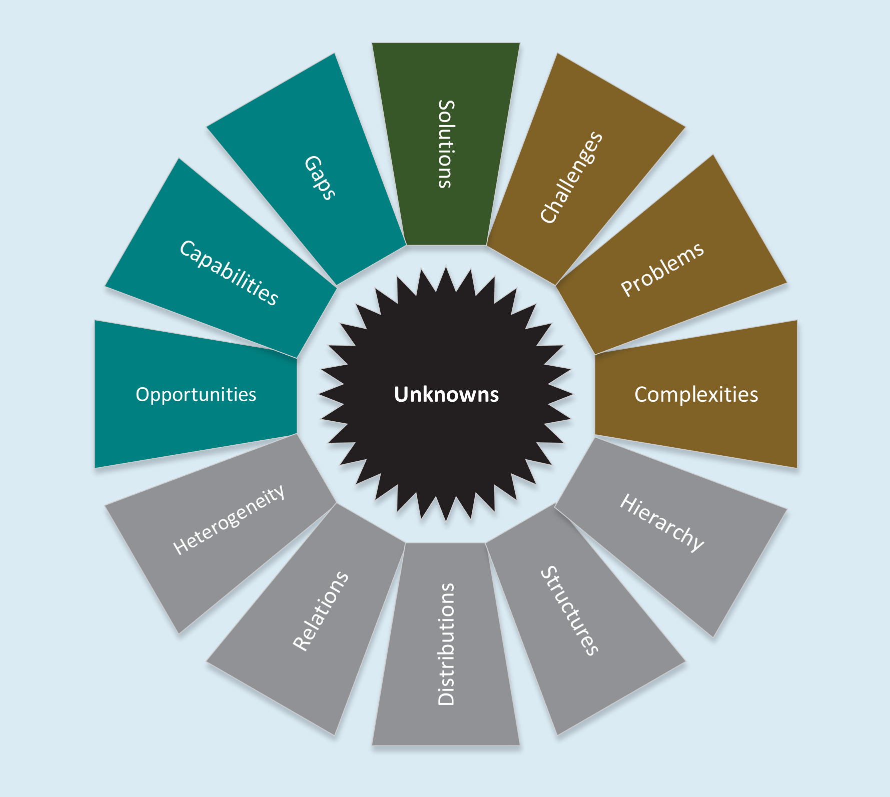
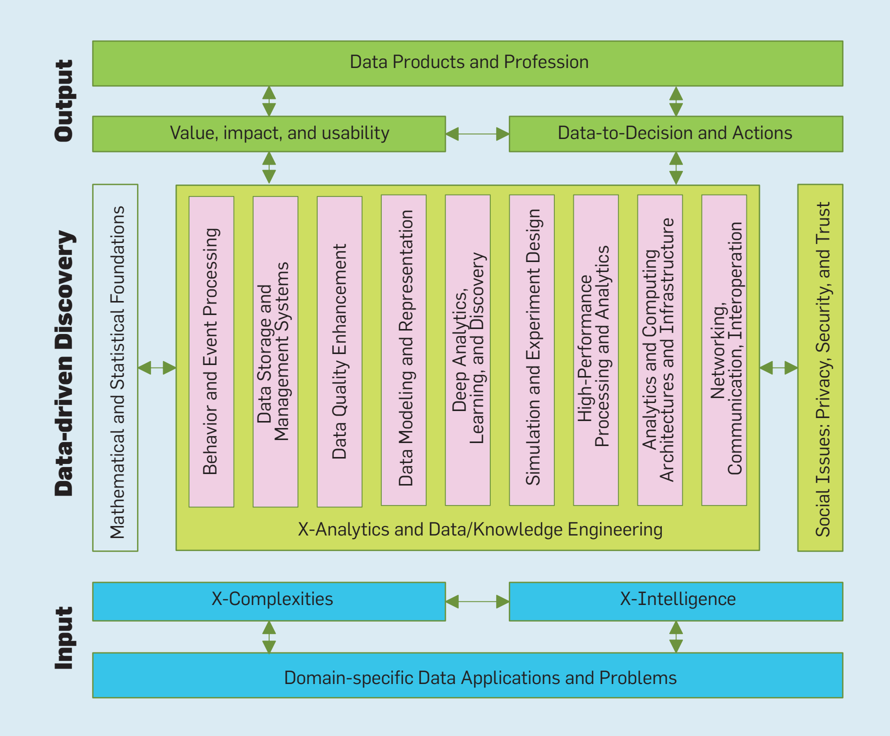


Figure 4. Data science conceptual landscape.


Output
Data Products and Profession
Value, impact, and usability
Data-to-Decision and Actions
Data-driven Discovery
Mathematical and Statistical Foundations
Behavior and Event Processing
Data Storage and Management Systems
Data Quality Enhancement
Data Modeling and Representation
Deep Analytics, Learning, and Discovery
Simulation and Experiment Design
High-Performance Processing and Analytics
Analytics and Computing Architectures and Infrastructure
Networking, Communication, Interoperation
X-Analytics and Data/Knowledge Engineering
Social Issues: Privacy, Security, and Trust
Input
X-Complexities
X-Intelligence
Domain-specific Data Applications and Problems

landscape of data science and its major research challenges by taking an interdisciplinary, complex-system-based, hierarchical view.

As in Figure 4, the data science landscape consists of three layers: "data input," including domain-specific data applications and systems, and X-complexity and X-intelligence in data and business problems; "data-driven discovery" consisting of discovery tasks and challenges; and "data output" consisting of results and outcomes.

Research challenges and opportunities emerge in all three in terms of five areas not otherwise managed well through non-data-science methodologies, theories, or systems:

Data/business understanding. The aim is for data scientists, as well as data users, to identify, specify, represent, and quantify the X-complexities and X-intelligence that cannot be managed well through existing theories and techniques but nevertheless are embedded in domain-specific data and business problems. Examples include how to understand in what forms, at what level, and to what extent the respective complexities and intelligence interact with one another and to devise methodologies and technologies for incorporating them into data science tasks and processes;

Mathematical and statistical foundation. The aim is to enable data scientists to disclose, describe, represent, and capture complexities and intelligence for deriving actionable insight. Existing analytical and computational theories may need to be explored as to whether, how, and why they are insufficient, missing, or problematic, then extended or redeveloped to address the complexities in data and business problems by, say, supporting multiple, heterogeneous, large-scale hypothesis testing and survey design, learning inconsistency, and uncertainty across multiple sources of dynamic data. Results might include deep representation of data complexities, large-scale, fine-grain personalized predictions, support for non-IID data learning, and creation of scalable, transparent, flexible, interpretable, personalized, parameter-free modeling;

The metasynthesis of X-complexities and X-intelligence in complex data science problems might ultimately produce even super machine intelligence.

Data/knowledge engineering and X-analytics. The aim is to develop domain-specific analytic theories, tools, and systems not available in the relevant body of knowledge to represent, discover, implement, and manage the data, knowledge, and intelligence and support the corresponding data and analytics engineering. Examples include automated analytical software that automates selection and construction of features and models, as well as the analytics process in its self-understanding of intrinsic data complexities and intelligence, and that self-monitors, self-diagnoses, and self-adapts to data characteristics, domain-specific context and learning objectives and potential, and learns algorithms that recognize data complexities and self-trains the corresponding optimal models customized for the data and objectives;

Data quality and social issues. The aim here is to identify, specify, and respect social issues in domain-specific data, business-understanding, and data science processes, including use, privacy, security, and trust, and make possible social issues-based data science tasks not previously handled well. Examples include privacy-preserving analytical algorithms and benchmarking the trustworthiness of analytical outcomes;

Data value, impact, utility. The aim is to identify, specify, quantify, and evaluate the value, impact, and utility of domain-specific data that cannot be addressed through existing measurement theories and systems. Examples involve data actionability, utility, and value; and

Data-to-decision and action-taking challenges. The aim is to develop decision-support theories and systems to enable data-driven decisions and insight-to-decision transformation, incorporating prescriptive actions and strategies that cannot be managed through existing technologies and systems. Examples include ways to transform analytical findings into decision-making strategies.

Since data/knowledge engineering and advanced analytics[6] play a key role in data science, the focus is on specific research questions not previously addressed. Data-quality enhancement is fundamental to handling data-quality issues like noise, uncertainty, missing values, and imbalance that may be present due to the increasing scale of complexity and data-quality issues (such as cross-organizational, cross-media, cross-cultural, and cross-economic mechanisms) emerging in the big-data and Internet-based data/business environment.

Data scientists seek to model, learn, analyze, and mine data, including X-complexities and X-intelligence. For example, being able to perform deep analytics is essential for discovering unknown knowledge and intelligence in the unknown space in Figure 2 that cannot be handled through existing latent learning and descriptive and predictive analytics; another option might be to integrate data-driven and model-based problem solving, balancing common learning models and frameworks and domain-specific data complexities and intelligence-driven evidence learning.

X-complexity and X-intelligence pose additional challenges to simulation and experimental design, including how to simulate the complexities, intelligence, working mechanisms, processes, and dynamics in data and corresponding business systems and how to design experiments to explore the effect of business managers' data-driven decisions. Big-data analytics requires high-performance processing and analytics that support large-scale, real-time, online, high-frequency, Internet-based, cross-organizational data processing and analytics while balancing local and global resource objectives. Such an effort may require new distributed, parallel, high-performance infrastructure, batch, array, memory, disk, and cloud-based processing and storage, data-structure-and-management systems, and data-to-knowledge management.

Complex data science also challenges existing analytics and computing architectures and infrastructure to, say, invent analytics and computing architectures and infrastructure based on memory, disk, cloud, and Internet resources. Another important issue for developers of data systems is how to support the networking, communication, and interoperation of the various data science roles within a distributed data science team. Such coordination requires distributed cooperative management of projects, data, goals, tasks, models, outcomes, work flows, task scheduling, version control, reporting, and governance.

Addressing them involves systematic and interdisciplinary approaches possibly requiring synergy among many related research areas. Such synergy is due to taking on complex data science problems that cannot be addressed through one-off efforts. For instance, data structures, computational infrastructure, and detection algorithms are required for high-frequency real-time risk analytics in extremely large online businesses like electronic commerce and financial trading.

Violating assumptions in data science. Big data includes X-complexities, including complex coupling relationships and/or mixed distributions, formats, types and variables, and unstructured and weakly structured data. Complex data poses significant challenges to many mathematical, statistical, and analytical methods built on relatively narrow assumptions, owing to the fact that they are routinely violated in big-data analytics. When assumptions are violated, modeling outcomes may be inaccurate, distorted, or misleading. In addition to general scenarios (such as whether data violates the assumptions of normal distribution, t-test, and linear regression), an assumption check applies to broad aspects of a business problem's data, including independence, normality, linearity, variance, randomization, and measurement that apply to population data and analysis.

Fundamental work on detecting and verifying such validations is limited, and even less has sought to invent new theories and tools to manage and circumvent assumption violations in big data. One such violation I highlight here is the IID assumption, as big, complex data (referring to objects, attributes, and values[2]) is essentially non-IID, whereas most existing analytical methods are IID.[2]

In a non-IID data problem (see Figure 5a), non-IIDness, as outlined in Figure 5c, refers to the mixture of couplings, including co-occurrence, neighborhood, dependence, linkage, correlation, and causality, and other poorly explored and unquantified relations involving, say, sophisticated cultural and religious connections and influence, as well as heterogeneity within and between two or more aspects of a data system (such as entity, entity class, entity property like a variable, process, fact, and state of affairs) or other types of entities or properties (such as learning algorithms, and learned results) appearing or produced prior to, during, or after a target process (such as a learning task). By contrast, IIDness essentially ignores or simplifies all these properties, as outlined in Figure 5b.

Learning visible and especially invisible non-IIDness is fundamental for data scientists looking for deep understanding of data with weak and/or un-

Figure 5. IIDness vs. non-IIDness in data science problems.

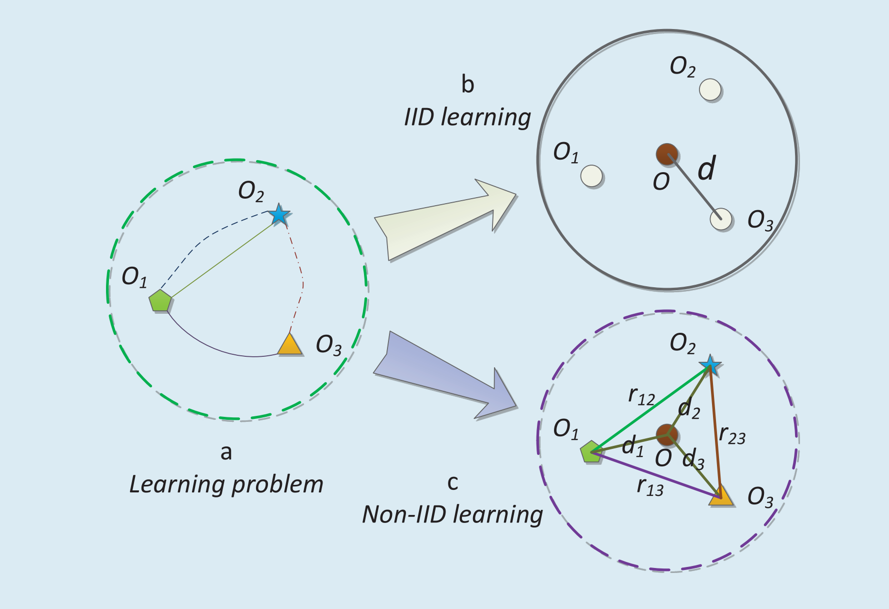


Figure 6. Synthesizing X-intelligence in data science.

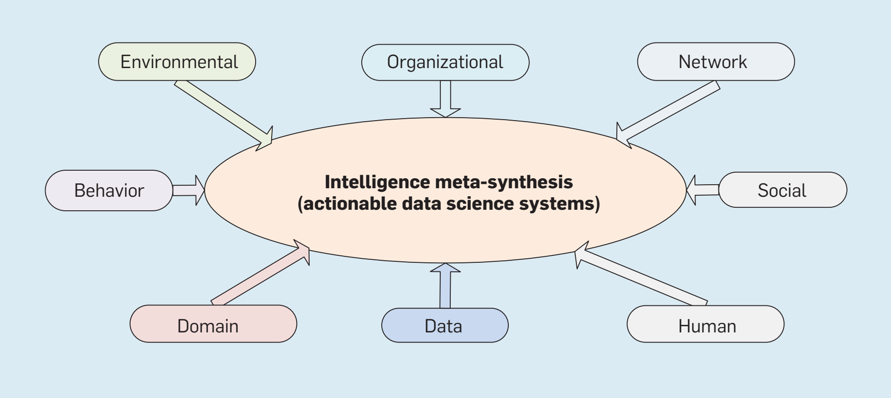
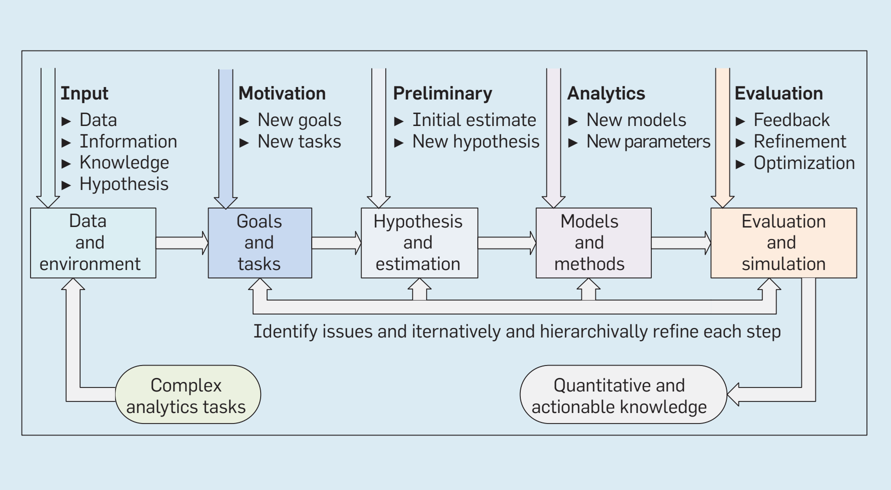


Figure 7. Complex data science problems: qualitative-to-quantitative X-intelligence metasynthesis.



clear structures, distributions, relationships, and semantics. In many cases, locally visible but globally invisible (or vice versa) non-IIDness takes a range of forms, structures, and layers on diverse entities. Individual learners cannot tell the whole story due to their inability to identify such complex non-IIDness. Effectively learning the widespread, visible, and invisible non-IIDness of big data is crucial for data scientists trying to gain a complete picture of an underlying business problem.

Data analysts often focus on learning explicit non-IIDness, or visible and easy to learn. The hybridization of multiple analytical methods on combinations of multiple sources of data into a big table for analysis typically falls into this category of non-IID systems. Computing non-IIDness refers to understanding, formalizing, and quantifying the non-IID aspects of data[2] (such as entities, interactions, layers, forms, and strength of non-IIDness). Non-IID learning systems seek to understand non-IIDness in data-analytics systems, from values, attributes, objects, methods, and measures to processing outcomes (such as mined patterns).

I now explore the prospects for inventing new data science theories and tools for non-IIDness and non-IID data learning,[2] including how to address non-IID data characteristics (not just variables), in terms of new feature analysis:

Deep understanding of non-IID data characteristics. The aim is to identify, specify, and quantify non-IID data characteristics, factors, types, and levels of non-IIDness in data and business, and identify the difference between what can be captured and what cannot be captured through existing technologies;

Non-IID feature analysis and construction. The aim is to invent new theories and tools for analyzing feature relationships by considering non-IIDness within and between features and objects and developing new theories and algorithms for selecting, mining, and constructing features;

Non-IID learning theories, algorithms, and models. The aim is to create new theories, algorithms, and models for analyzing, learning, and mining non-IID data by considering various couplings and heterogeneity; and

Non-IID similarity and evaluation metrics. The aim is to develop new similarity and dissimilarity learning methods and metrics, as well as evaluation metrics that consider non-IIDness in data and business.

More broadly, many existing data-oriented theories, designs, mechanisms, systems, and tools may have to be reinvented in light of non-IIDness. In addition to incorporating non-IIDness into data mining, machine learning, and general data analytics, non-IIDness is found in other well-established bodies of knowledge, including mathematical and statistical foundations, descriptive-analytics theories and tools, data-management theories and systems, information-retrieval theories and tools, multimedia analysis, and various X-analytics.[6]

Data characteristics and X-complexities. To address critical issues in data-driven discovery like assumption violations, I assume data characteristics and X-complexities determine the values, complexities, and quality of data-driven discovery. Data characteristics refer to the profile and complexities of data (generally a dataset) that can be described in terms of data factors (such as distribution, structure, hierarchy, dimension, granularity, heterogeneity, and uncertainty).

Understanding data characteristics and X-complexities involves four fundamental data science challenges and directions:[6] definition of data characteristics and X-complexities; how to represent and model data characteristics and X-complexities; data-characteristics- and X-complexities-driven data understanding, analysis, learning, and management; and how to evaluate the quality of data understanding, analysis, learning, and management in terms of data characteristics and X-complexities. Unfortunately, only limited theories and tools are available for addressing them.

Data-brain and human-like ma-

chine intelligence. Computer scientists, economists, and politicians, as well as the general public, debate whether and when machines might replace humans.[22] While it may not be possible to build data-brain or thinking machines with human-like abilities, data science, especially big-data analytics, is driving a technological revolution, from implementing logical-thinking-centered machine intelligence to creative-thinking-oriented machine intelligence. It may be partially reflected in Google's AlphaGo (https://deepmind.com/) defeat of top-ranked Chinese Go player Ke Jie in 2017 and South Korean grandmaster Lee Sedol in 2016, as well as the Facebook emotion experiment,[14] but none has actually exhibited human-like imagination or thinking. This revolution (such as through data science thinking[6]), if truly able to mimic human intelligence, may transform machine intelligence, changing the human-machine separation of responsibilities.

Curiosity is a critical human capability, starting the moment we are born. We want to know what, how, and why everything. Curiosity connects other cognitive activities, particularly imagination, reasoning, aggregation, creativity, and enthusiasm to produce new ideas, observations, concepts, knowledge, and decisions. Humans manage to upgrade their own intelligence through experience, exploration, learning, and reflection. Accordingly, a critical goal for data scientists is to enable data- and X-intelligence-driven machines to generate, retain, and simulate human curiosity through learning inquisitively from data and X-intelligence.

Imaginative thinking differentiates humans from machines designed with sense-effect, learning, reasoning, and optimization mechanisms. Human imagination is intuitive, creative, evolving, and uncertain. It also represents a great yet challenging opportunity for transforming logic, patterns, and predefined sense-effect-mechanisms-driven machines into human-like data systems. Such a transformation would require machines able to simulate human-imagination processes and mechanisms. Existing knowledge representation, aggregation, computational logic, reasoning, and logic thinking incorporated into machines may never quite deliver machine curiosity, intuition, or imagination. Existing data and computer theories, operating systems, system architectures and infrastructures, computing languages, and data management must still be fundamentally reformed by, say, simulating, learning, reasoning, and synthesizing original thoughts from cognitive science, social science, data science, and intelligence science to render machines creative. They also must be able to engage X-intelligence in a non-predefined, "non-patternable" way, unlike existing simulation, learning, and computation, which are largely predefined or design-based by default.

To enable discovery, data-analytical thinking, a core aspect of data science thinking,[6] needs to be built into data products and learned by data professionals. Data-analytical thinking is not only explicit, descriptive, and predictive but also implicit and prescriptive. Complex data problem solving requires systematic, evolving, imaginative, critical, and actionable data science thinking. In addition to computational thinking, a machine might ultimately be able to mimic human approaches to information processing by synthesizing comprehensive data, information, knowledge, and intelligence through cognitive-processing methods and processes.

Developing Complex Systems

The X-complexities and X-intelligence discussed earlier render a complex data system equivalent to an open complex intelligent system.[3] Use of X-intelligence by a data scientist could take one of two paths: "single intelligence engagement" or "multi-aspect intelligence engagement." An example of the former is domain knowledge in data analytics and user preferences in recommender systems. Single-intelligence engagement applies to simple data science problem solving and systems. In general, multi-aspect X-intelligence can be found in complex data science problems.

As outlined in Figure 6, the performance of a data science-problem-solving system depends on recognition, acquisition, representation, and integration of relevant X-intelligence. To enable an X-intelligence-driven complex data science problem-solving process, data scientists need new methodologies and system-engineering methods. The theory of "metasynthetic engineering"[3,20] and integration of ubiquitous intelligence might provide useful guidance for synthesizing X-intelligence in complex data and business systems.

The principle of "intelligence metasynthesis" of multiple types of intelligence[3,20] involves, synthesizes, and uses ubiquitous intelligence in the complex data environment to understand the nature of data and related problems, invent discovery systems, discover interesting knowledge, and generate actionable insights.[7] Intelligence metasynthesis applies to solving complex data science problems involving complex system engineering in which multiple aspects of complexity and intelligence may be embedded in the data, environment, and problem-solving process. The "reductionism" methodology[3] for data and knowledge exploration may not work well because the problem may not be clear, specific, and quantitative so cannot be decomposed and analyzed effectively. In contrast, analysis through a holistic lens does not equal the sum of the analysis of the parts, a common challenge developing complex systems.[20]

Accordingly, in light of the theory of "system complexities" and corresponding methodologies "systematism,"[3,20] a methodology that synthesizes reductionism and "holism"[6,7] may then be more applicable for analyzing, designing, and evaluating complex data problems.

When a data science problem involves large-scale data objects, multiple levels of subtasks, objects, sources, and types of data from online, business, mobile, or social networks, complicated contexts, human involvement and domain constraints, the problem thus reflects the characteristics of an open complex system.[3,20] The problem is also likely to involve common system complexities, including openness, scale, hierarchy, human involvement, societal characteristics, dynamic characteristics, uncertainty, and imprecision.[3,19,20]

Although specific big-data analytical tasks are manageable by following existing analytical methodologies, typical cross-enterprise, global, and Internet-based data science projects (such as global financial crisis and terrorist activities) satisfy most if not all such complexities. This level of complex data science involves X-complexities problems, and their resolution must first synthesize the X-intelligence in the problems. One approach to instantiate the systematism methodology is "qualitative-to-quantitative metasynthesis,"[3,20] as proposed by Chinese scientist Xuesen Qian (also known as Hsue-Shen Tsien) to guide system engineering in large-scale open systems.[20] Such qualitative-to-quantitative metasynthesis supports exploration of open complex systems through an iterative cognitive and problem-solving process on a human-centered, human-machine-cooperative problem-solving platform in which human, data, and machine intelligence, along with X-intelligence, must be engaged, quantified, and synthesized. Implementing it for open complex intelligent systems, the "metasynthetic computing and engineering" (MCE) approach[3] provides a systematic computing and engineering guide and suite of system-analysis tools.

Figure 7 outlines the process of applying the qualitative-to-quantitative metasynthesis methodology to complex data science problems. MCE supports an iterative, hierarchical problem-solving process, incorporating internal and external inputs, including data, information, domain knowledge, initial hypotheses, and underlying environmental factors. Data scientists would start by presetting analytics goals and tasks to be explored on the given data by incorporating domain, organizational, social and environmental complexities and intelligence. They would then use preliminary observations obtained from domain and experience to identify and verify qualitative and quantitative hypotheses and estimations that guide development of modeling and analytics methods. Findings would then be evaluated and fed back to the corresponding procedures for refining and optimizing understanding of previously unknown problem challenges, goals, and discovery methods. Following these iterative and hierarchical steps toward qualitative-to-quantitative intelligence transformation would thus disclose and quantify the initial problem "unknownness." Finally, actionable knowledge and insight would be identified and delivered to businesspeople who would address data complexities and business goals.

As an example of how to deliver actionable knowledge, domain-driven data mining[7] aims to integrate X-intelligence and X-complexities for complex knowledge-discovery problems. Domain-driven data mining advocates a comprehensive process of synthesizing data intelligence with other types of intelligence to prompt new intelligence to address gaps in existing data-driven methods, delivering actionable knowledge to business users. The metasynthesis of X-complexities and X-intelligence in complex data science problems might ultimately produce even super machine intelligence. Super-intelligent machines could then understand, represent, and learn X-complexities, particularly data characteristics; acquire and represent unstructured, ill-structured, and uncertain human knowledge; support involvement of business experts in the analytics process; acquire and represent imaginative and creative thinking in group heuristic discussions among human experts; acquire and represent group/collective interaction behaviors; and build infrastructure involving X-intelligence. While a data brain cannot mimic special human imagination, curiosity, and intuition, the simulation and modeling of human behavior and human-data systems interaction and cooperation promise to approach human-like machine intelligence.

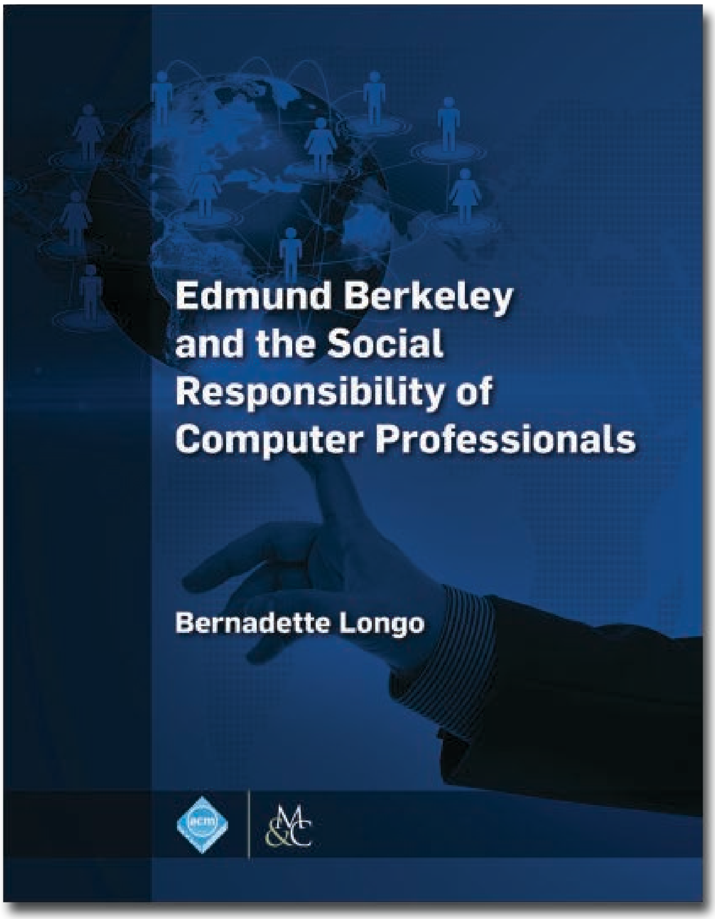
Conclusion

The low-level X-complexities and X-intelligence characterizing complex data science problems reflect the gaps between the world of hidden data and existing data science immaturity. Filling them requires a disciplinewide effort to build complex data science thinking and corresponding methodologies from a complex-system perspective. The emerging data science evolution means opportunities for breakthrough research, technological innovation, and a new data economy. If parallels are drawn between evolution of the Internet and evolution of data science, the future and the socioeconomic and cultural impact of data science will be unprecedented indeed, though as yet unquantifiable.

References

1. Cao, L.B. In-depth behavior understanding and use: The behavior informatics approach. *Information Science 180*, 17 (Sept. 2010), 3067–3085.
2. Cao, L.B. Non-IIDness learning in behavioral and social data. *The Computer Journal 57*, 9 (Sept. 2014), 1358–1370.
3. Cao, L.B. *Metasynthetic Computing and Engineering of Complex Systems*. Springer-Verlag, London, U.K., 2015.
4. Cao, L.B. Data science: Nature and pitfalls. *IEEE Intelligent Systems 31*, 5 (Sept.-Oct. 2016), 66–75.
5. Cao, L.B. Data science: A comprehensive overview. *ACM Computing Surveys* (to appear).
6. Cao, L.B. *Understanding Data Science*. Springer, New York (to appear).
7. Cao, L.B., Yu, P.S., Zhang, C., and Zhao, Y. *Domain Driven Data Mining*. Springer, Springer-Verlag, New York, 2010.
8. Cao, L.B., Yu, P.S., and Kumar, V. Nonoccurring behavior analytics: A new area. *IEEE Intelligent Systems 30*, 6 (Nov. 2015), 4–11.
9. Cleveland, W.S. Data science: An action plan for expanding the technical areas of the field of statistics. *International Statistical Review 69*, 1 (Dec. 2001), 21–26.
10. Diggle, P.J. Statistics: A data science for the 21st century. *Journal of the Royal Statistical Society: Series A (Statistics in Society) 178*, 4 (Oct. 2015), 793–813.
11. Donoho, D. *50 Years of Data Science*. Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, 2015; http://courses.csail.mit.edu/18.337/2015/docs/50YearsDataScience.pdf
12. Huber, P.J. *Data Analysis: What Can Be Learned from the Past 50 Years*. John Wiley & Sons, Inc., New York, 2011.
13. Jagadish, H., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J.M., Ramakrishnan, R., and Shahabi, C. Big data and its technical challenges. *Commun. ACM 57*, 7 (July 2014), 86–94.
14. Kramer, A.D., Guillory, J.E., and Hancock, J.T. Experimental evidence of massive-scale emotional contagion through social networks. *Proceedings of the National Academy of Sciences 111*, 24 (Mar. 2014), 8788–8790.
15. Lazer, D., Kennedy, R., King, G., and Vespignani, A. The parable of Google flu: Traps in big data analysis. *Science 343*, 6176 (Mar. 2014), 1203–1205.
16. Manyika, J. and Chui, M. *Big Data: The Next Frontier for Innovation, Competition, and Productivity*. McKinsey Global Institute, 2011.
17. Matsudaira, K. The science of managing data science. *Commun. ACM 58*, 6 (June 2015), 44–47.
18. Mattmann, C.A. Computing: A vision for data science. *Nature 493*, 7433 (Jan. 24, 2013), 473–475.
19. Mitchell, M. *Complexity: A Guided Tour*. Oxford University Press, Oxford, U.K., 2011.
20. Qian, X., Yu, J., and Dai, R. A new discipline of science—The study of open complex giant system and its methodology. *Journal of Systems Engineering and Electronics 4*, 2 (June 1993), 2–12.
21. Rowley, J. The wisdom hierarchy: Representations of the DIKW hierarchy. *Journal of Information and Communication Science 33*, 2 (Apr. 2007), 163–180.
22. Suchma, L. *Human-Machine Reconfigurations: Plans and Situated Actions*. Cambridge University Press, Cambridge, U.K., 2006.
23. Tukey, J.W. The future of data analysis. *The Annals of Mathematical Statistics 33*, 1 (Mar. 1962), 1–67.
24. Tukey, J.W. *Exploratory Data Analysis*. Pearson, 1977.

Longbing Cao (longbing.cao@gmail.com) is a professor in the Advanced Analytics Institute at the University of Technology Sydney, Australia.


Edmund Berkeley
and the Social
Responsibility of
Computer Professionals
Bernadette Longo
acm
M&C

DOI:10.1145/3107239

Mathematics solves problems by pen and paper. CS helps us to go far beyond that.

BY MARIJN J.H. HEULE AND OLIVER KULLMANN

The Science of Brute Force

RECENT PROGRESS IN automated reasoning and supercomputing gives rise to a new era of brute force. The game changer is "SAT," a disruptive, brute-reasoning technology in industry and science. We illustrate its strength and potential via the proof of the Boolean Pythagorean Triples Problem, a long-standing open problem in Ramsey Theory. This 200TB proof has been constructed completely automatically—paradoxically, in an ingenious way. We welcome these bold new proofs emerging on the horizon, beyond human understanding—both mathematics and industry need them.

Many relevant search problems, from artificial intelligence to combinatorics, explore large search spaces to determine the presence or absence of a certain object. These problems are hard due to combinatorial explosion, and have traditionally been called infeasible. The brute-force method, which at least implicitly explores all possibilities, is a general approach to systematically search through such spaces.

Brute force has long been regarded as suitable only for simple problems. This has changed in the last two decades, due to the progress in Satisfiability (SAT) solving, which by adding brute reason renders brute force into a powerful approach to deal with many problems easily and automatically. Search spaces with far more possibilities than the number of particles in the universe may be completely explored.

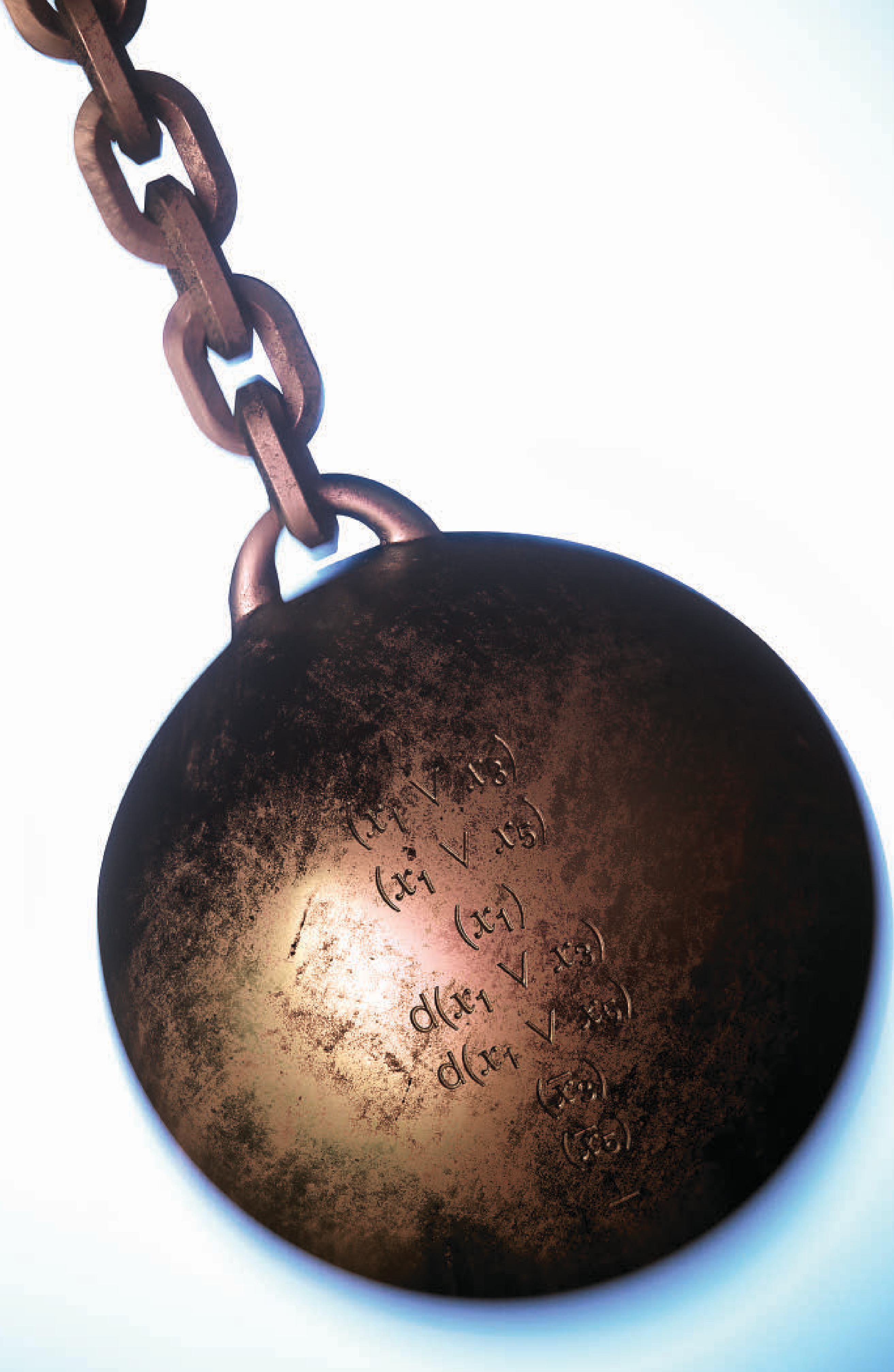
SAT solving determines whether a formula in propositional logic has a solution, and its brute reasoning acts in a blind and uninformed way—as a feature, not a bug. We focus on applying SAT to mathematics, as a systematic development of the traditional method of proof by exhaustion.

Can we trust the result of running complicated algorithms on many machines for a long time? The strongest solution is to provide a proof, which is also needed to show correctness of highly complex systems, which are everywhere, from finance to health care to aviation.

» key insights

- Long-standing open problems in mathematics can now be solved completely automatically resulting in clever though potentially gigantic proofs.
- Our time requires answers to hard questions regarding safety and security. In these cases knowledge is more important than understanding as long as we can trust the answers.
- Powerful SAT-solving heuristics facilitate linear speedups even when using thousands of cores. Combined with the ever-increasing capabilities of high-performance computing clusters they enable solving challenging problems.

ILLUSTRATION BY PETER CROWTHER ASSOCIATES


$(x_1 \vee x_3)$
$(x_1 \vee x_5)$
(x_1)
$d(x_1 \vee x_3)$
$d(x_1 \vee x_5)$
$(\bar{x}_3)$
$(\bar{x}_5)$

Many problems arising from areas such as Ramsey Theory and formal methods appear to be intrinsically hard and may be only solvable by SAT. Any proof for such problems may be huge, in which case mathematicians will not be able to produce a paper proof. The enormous size of such proofs hardly influences confidence in the correctness, as highly trusted systems can validate them.

We argue that obtaining such results is meaningful regardless of our ability to understand them.

The Rise of Brute Force

We all know that brute force does not work, or at least is brutish, do we not? In our case it is even "brute reasoning."

> ***I can stand brute force, but brute reason is quite unbearable. There is something unfair about its use. It is hitting below the intellect.***
>
> O. Wilde

A mathematician using "brute force" is a kind of barbaric monster, is she not? Case distinctions play an important role for thinking, but if the number of cases gets too big, it seems impossible to obtain an overview, and one has to slavishly follow the details. But perhaps this is what our times demand?

In the beginning of the 20th century there was a very optimistic outlook for mathematics. Gödel's Incompleteness Theorem seemed to destroy the positive spirit of the time, famously expressed by Hilbert's "We must know. We will know." That said, even Gödel anticipated the relevance of SAT solving in his letter to von Neumann[a], shifting the attention to finitizing infinite problems. Today, SAT solving on high-performance computing systems enables us to conquer problems of high complexity, driven by practice. This combination of enormous computational power with "magical brute force" can now solve very hard combinatorial problems, as well as proving safety of systems such as railways.

Our guiding example is the *Pythagorean Triples Problem*,[15,25] a typical problem from Ramsey Theory: we consider all partitions of the set $\{1, 2, \ldots\}$ of natural numbers into finitely many parts, and the question is whether always at least one part contains a Pythagorean triple (a, b, c) with $a^2 + b^2 = c^2$. For example when splitting into odd and even numbers, then the odd part does not contain a Pythagorean triple (due to odd plus odd = even), but the even part contains for example $6^2 + 8^2 = 10^2$. We show that the answer is *yes*,[15] when partitioning into two parts, and we conjecture the answer to be *yes* for any finite size of the partition.

To solve the *Boolean Pythagorean Triples Problem*, it suffices to show the existence of a subset of the natural numbers, such that any partition of that subset into two parts has one part containing a Pythagorean triple. We focus on subsets $\{1, \ldots, n\}$, and determined by SAT solving that the smallest n for which the property holds is 7825. Plain brute force cannot help, since 2^{7825}, the number of possible partitions into two parts, is way too big. So really "clever" algorithms are needed. An interesting aspect here is that there is no known ordinary mathematical existence proof for any form of the Pythagorean Triples Problem, even when generalizing the problem from triples $a^2 + b^2 = c^2$ to tuples $t_1^2 + \cdots + t_{k-1}^2 = t_k^2$. Only computational proofs are known and, so far at least, only SAT solving can deal with the harder problems. We show that $\{1, \ldots, 10^7\}$ can be partitioned into *three* parts, such that *no part* contains a Pythagorean triple. Thus if there is an n such that every 3-partitioning of $\{1, \ldots, n\}$ has a part containing a Pythagorean triple, then $n > 10^7$. Due to this enormous size, it is thus conceivable that the truth of the *three-valued Pythagorean Triples Problem* might never be known.

Before considering the solution process, one may ask, why should we care? Are there problems, for which such reasoning is really useful? Yes, the same techniques are used to prove correctness of hardware and software systems. Finding a bug in a large hardware system is essentially the same as finding a counter-example, and thus is similar to finding a partition avoiding all Pythagorean triples. *Proving* correctness of a system, that is, there is no counter-example, is similar to proving that each partition must contain some Pythagorean triple. SAT solving has revolutionized hardware verification,[5] and now SAT can come to the rescue of mathematics, solving very hard combinatorial problems previously completely out of reach. This collaboration works in both directions, as the applications in mathematics, especially Ramsey Theory, sharpen SAT algorithms: the Cube-and-Conquer method[16] was developed for computing van der Waerden numbers,[1] and recently the Cube-and-Conquer solver TREENGELING[b] won the parallel track of the 2016 SAT competition.[c] Deeper mathematical investigations into the structure of the SAT instances could help with understanding and improving SAT in general.

Well known early mathematical proofs using *Proof by Exhaustion* are the Four-Color Theorem[37] and the proof that no projective plane of order 10 exists.[24] The former is actually a rather small case-distinction by modern standards (only hundreds of cases). The latter invokes a larger, but also man-made case-split (billions of cases), for which it can be determined in advance whether this will succeed. In contrast, we have currently no way of knowing whether the SAT solver's "magic" is sufficient to solve a given problem.

Throughout this article we use the *Boolean Schur Triple Problem* as an example: does there exists a red/blue coloring of the numbers 1 to n, such that there is no monochromatic solution of $a + b = c$ with $a < b < c \leq n$. Compared to the Boolean Pythagorean Triples Problem, all natural numbers are involved, not just square numbers. As a result, there are many more triples, and unsatisfiability is reached much sooner. For $n = 8$ such a coloring exists: color the numbers 1, 2, 4, 8 red and 3, 5, 6, 7 blue. However such a coloring is not possible for $n = 9$. A naive brute-force algorithm would consider all $2^9 = 512$ possible red/blue colorings. We will show that with brute reasoning only six (or even four) red/blue colorings need to be evaluated.

The Art of SAT Solving

A SAT problem uses Boolean variables v (they can be assigned to either `true` or `false`), which are constrained using

[a] https://rjlipton.wordpress.com/the-gdel-letter/.

[b] http://fmv.jku.at/lingeling/.

[c] http://www.satcompetition.org/.

clauses, which are disjunctions of literals x. Literals are either variables $x = v$ or their negations $x = \bar{v}$. A literal x (or $\bar{x}$) is `true` if the corresponding variable v is assigned to `true` (or `false`, respectively). A clause is satisfied if at least one of its literals is assigned to `true`. A SAT formula is a conjunction of clauses. We refer to a solution of a SAT formula as an assignment to its variables that satisfies all its clauses. Formulas with a solution are called *satisfiable*, while formulas without solutions are called *unsatisfiable*. Let $\vee$ and $\wedge$ refer to the logical OR and AND operators, respectively. For example, the formula $(x \vee \bar{y}) \wedge (\bar{x} \vee y)$ with two clauses is satisfiable. The solutions for this formula are the two assignments that assign both x and y to the same value.

SAT solvers, programs that solve SAT formulas, have become extremely powerful over the last two decades. Progress has been by leaps and bounds, starting with the pioneering work by Davis and Putnam until the early 1990s when solvers could handle formulas with thousands of clauses. Today's solvers can handle formulas with millions of clauses. This performance boost resulted in the *SAT revolution*:[3] encode problems arising from many interesting applications as SAT formulas, solve these formulas, and decode the solutions to obtain answers for the original problems. This is in a sense just using the *NP-completeness* of SAT:[6,11,19] every problem with a notion of "solution"—where these solutions are relatively short and where an alleged solution can be verified (or rejected) quickly—can be reduced to SAT efficiently. For many years NP-completeness was used only as a sign of "you can not solve it!", but the SAT revolution has put this back on its feet. For many applications, including hardware and software verification,[7, 18] SAT solving has become a disruptive technology that allows problems to be solved faster than by other known means.

The main paradigms of SAT solving are the incomplete *local search*,[20] which can only find satisfying assignments, and the two complete paradigms (which can also determine unsatisfiability), *look-ahead*[17] and *Conflict-Driven Clause Learning*[28] (CDCL). Local search tries to find a solution via local modifications to total assignments (using all variables). Look-ahead recursively splits the problem as cleverly as possible into subproblems, via looking-ahead. CDCL tries to assign variables to find a satisfying assignment in a straight-forward way, and if that fails (the normal case), then the failure is transformed into a clause, which is added to the formula. Here, we first explain CDCL, which is mainly responsible for the SAT revolution. Afterwards we describe how look-ahead can enhance CDCL on hard problems.

CDCL SAT solving algorithms cycle through three phases: *simplify*, *decide*, and *learn*. Solvers maintain an assignment (initially empty) and each phase updates that assignment. During *simplify* the assignment is extended by detecting new inferences. Afterwards, *decide* heuristically picks an unassigned variable and assigns it to `true` or `false`. After iterating these two phases, the current assignment either satisfies the formula, which terminates the search, or falsifies a clause. In the latter case, *learn* this conflict, as a clause, and modify the assignment to resolve the conflict. If the empty clause $\perp$ is learned, the solver detects unsatisfiability, otherwise simplify-decide is performed again, etc. Look-ahead differs from CDCL by using stronger means for *simplify* and *decide*, but weaker means for *learn*.

The most basic inference mechanism in SAT solvers works as follows: a clause is *unit* under an assignment that falsifies all but one of its literals, while leaving the remaining literal unassigned. The only possibility to satisfy a unit clause (under that assignment) is to assign the remaining literal to `true`. A key SAT solving technique is *Unit Clause Propagation* (UCP): Given an assignment and a formula, while the formula has unit clauses, extend the assignment by satisfying the remaining literals in the unit clauses. UCP has two possible terminating states: either all unit clauses have been satisfied, or there is a falsified clause due to two complementary unit clauses (x) and $(\bar{x})$. In the latter case, we say that UCP results in a *conflict*. Conflicts are analyzed to obtain new clauses. These *conflict clauses* are added to the formula to prevent the solver from visiting that assignment in the future. Additionally, conflict analysis updates the heuristics to guide the solver towards a short refutation.

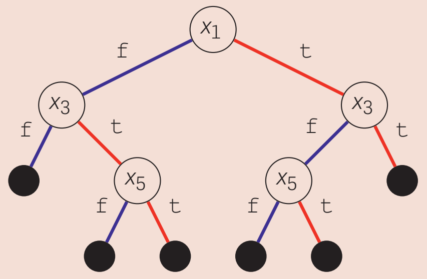
There are two types of decision heuristics for SAT solvers: *focus* and *global* heuristics. Focus heuristics, also known as conflict-driven heuristics (for CDCL solvers), aim at finding short refutations. These heuristics are cheap to compute and have been highly successful in solving large problems arising from industrial applications. In short, focus heuristics work as follows: whenever a solver encounters a conflicting state, the importance of the variables that cause the conflict is increased. Simply making these variables more

Figure 1. Encoding and case split of Boolean Schur Triples Problem.

Encoding

$$
\begin{aligned}
&(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4) \wedge\\
&(x_1 \vee x_4 \vee x_5) \wedge (\bar{x}_1 \vee \bar{x}_4 \vee \bar{x}_5) \wedge (x_2 \vee x_3 \vee x_5) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_5) \wedge\\
&(x_1 \vee x_5 \vee x_6) \wedge (\bar{x}_1 \vee \bar{x}_5 \vee \bar{x}_6) \wedge (x_2 \vee x_4 \vee x_6) \wedge (\bar{x}_2 \vee x_4 \vee \bar{x}_6) \wedge\\
&(x_1 \vee x_6 \vee x_7) \wedge (\bar{x}_1 \vee \bar{x}_6 \vee \bar{x}_7) \wedge (x_2 \vee x_5 \vee x_7) \wedge (\bar{x}_2 \vee \bar{x}_5 \vee \bar{x}_7) \wedge\\
&(x_3 \vee x_4 \vee x_7) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_7) \wedge (x_1 \vee x_7 \vee x_8) \wedge (\bar{x}_1 \vee \bar{x}_7 \vee \bar{x}_8) \wedge\\
&(x_2 \vee x_6 \vee x_8) \wedge (\bar{x}_2 \vee \bar{x}_6 \vee \bar{x}_8) \wedge (x_3 \vee x_5 \vee x_8) \wedge (\bar{x}_3 \vee \bar{x}_5 \vee \bar{x}_8) \wedge\\
&(x_1 \vee x_8 \vee x_9) \wedge (\bar{x}_1 \vee \bar{x}_8 \vee \bar{x}_9) \wedge (x_2 \vee x_7 \vee x_9) \wedge (\bar{x}_2 \vee \bar{x}_7 \vee \bar{x}_9) \wedge\\
&(x_3 \vee x_6 \vee x_9) \wedge (\bar{x}_3 \vee \bar{x}_6 \vee \bar{x}_9) \wedge (x_4 \vee x_5 \vee x_9) \wedge (\bar{x}_4 \vee \bar{x}_5 \vee \bar{x}_9)
\end{aligned}
$$

Case split as binary tree



important than all the other variables results in state-of-the-art performance on most industrial problems.[2]

If no short refutation exists (or is too hard to find), it is best to use global heuristics (for look-ahead solvers) to split the search space into two parts that are both easier to solve. Global heuristics are based on *look-aheads*:[23] for a given formula F, a look-ahead on literal x assigns x to `true`, applies UCP, and computes the set S of clauses in F that are shortened, but not satisfied. The heuristic value of a look-ahead on x is based on a weighted sum of the clauses in S, where clause weights depend on the length of clauses.

Both focus and global heuristics can reduce the search space exponentially. For really hard problems, such as the Pythagorean Triples Problem, it is best to combine both types of heuristics. Focus heuristics are effective when there exists a short refutation of the formula. For hard problems, initially there are no short refutations. One therefore needs to partition such a problem using global heuristics until the short refutations manifest themselves. This is the main idea behind the Cube-and-Conquer SAT solving paradigm,[16] which was crucial to solve the Pythagorean Triples Problem.

Consider again the Boolean Schur Triples Problem on the existence of a red/blue coloring of 1, …, 9 without a monochromatic solution of $a + b = c$. Figure 1 shows the SAT encoding, consisting of 32 clauses using the Boolean variables x_1, …, x_9. If variable x_i is assigned to `true` (`false`), then number i is colored red (blue). For each of the 16 solutions of $a + b = c$, there are two clauses: one stating that at least one of a, b, or c must be colored red, one stating that at least one of them must be colored blue. A binary tree is shown right beside the clauses. Each internal node contains a splitting variable x_i. The left branches assign decision variables to `false` (blue edge), while the right branches assign decision variables to `true` (red edge). Each leaf node represents an assignment that would result in a conflict during UCP. For example, for the left-most leaf node, x_1 and x_3 are assigned to `false` (blue): thus x_2, x_4 have to be set to `true` (due to $1 + 2 = 3$ and $1 + 3 = 4$), forcing x_6 to `false` ($2 + 4 = 6$), which forces x_7 and x_9 to `true` ($1 + 6 = 7$ and $3 + 6 = 9$), which yields the conflict $2 + 7 = 9$ with all three set to `true` (red). This node matches the first clause in the proof of Figure 2. The binary tree (a simple form of look-ahead solving) illustrates that heuristics can reduce the number of assignments to be evaluated from 512 to 6.

Due to the limited size of the example formula, relatively simple heuristics are sufficient to reduce the number of cases from 512 to 6. One such simple heuristic is Maximum Occurrences in clauses of Minimal Size (MOMS). Initially, all clauses are ternary and variable x_1 occurs most frequently. Therefore x_1 is used as the first decision variable. After simplification, several variables occur most frequently in binary clauses (twice), but variable x_3 has the best tie break (occurrences in remaining ternary clauses). Therefore variable x_3 is the best decision on the second level of the tree. Finally, variable x_5 is the most occurring variable in binary clauses on the third level.

A crucial aspect of solving the Boolean Pythagorean Triples Problem was the use of a dedicated look-ahead heuristic based on the recursive weight heuristic for random 3-SAT formulas. The three magic constants in this heuristic have been manually tweaked to achieve strong performance on the Boolean Pythagorean Triples Problem.[15] We estimate that the use of this optimized look-ahead heuristic reduced the number of cases by at least two orders of magnitude compared to alternative heuristics, such as focus heuristics or MOMS. Look-ahead heuristics were popular in the 1990s, but they have been mostly ignored after CDCL emerged. The usefulness of look-ahead heuristics to boost the performance on hard problems may revive the interest.

Proofs of Unsatisfiability

The unpredictable effectiveness of SAT solvers, together with their non-trivial implementations (needed for real-world efficiency), raise the question of whether their results can be trusted. If a problem has a solution, it is easy to verify that the given solution is correct: simply check whether the solution satisfies at least one literal in every clause. However, a claim that no solution exists is much harder to validate. Since SAT solvers use many complicated techniques that could result in implementation as well as conceptual errors, a method is required to verify unsatisfiability claims.

There are two approaches to deal with the trust issue of complicated software: prove its correctness or produce a certificate which can be validated with a simple program. Work in the first direction resulted in verified SAT solving.[31] However, this approach has two disadvantages: only some state-of-the-art techniques are verified, and verification is performed only on "higher levels," and thus excludes the low-level implementation tricks that

Figure 2. Proof and unit clause justification of the Boolean Schur Triples Problem.

Proof	Unit clause justification
$(x_1 \lor x_3)$	$(\cancel{x_1} \lor x_2 \lor \cancel{x_3})$, $(\cancel{x_1} \lor \cancel{x_3} \lor x_4)$, $(\cancel{\bar{x}_2} \lor \cancel{\bar{x}_4} \lor \bar{x}_6)$, $(\cancel{x_1} \lor \cancel{x_6} \lor x_7)$, $(\cancel{x_3} \lor \cancel{x_6} \lor x_9)$, $(\cancel{\bar{x}_2} \lor \cancel{\bar{x}_7} \lor \cancel{\bar{x}_9})$
$(x_1 \lor x_5)$	$(\cancel{x_1} \lor x_3)$, $(\cancel{x_1} \lor x_4 \lor \cancel{x_5})$, $(\cancel{x_1} \lor \cancel{x_5} \lor x_6)$, $(\bar{x}_2 \lor \cancel{\bar{x}_4} \lor \cancel{\bar{x}_6})$, $(\cancel{x_2} \lor \cancel{x_5} \lor x_7)$, $(\cancel{\bar{x}_3} \lor \cancel{\bar{x}_4} \lor \cancel{\bar{x}_7})$
(x_1)	$(\cancel{x_1} \lor x_3)$, $(\cancel{x_1} \lor x_5)$, $(\bar{x}_2 \lor \cancel{\bar{x}_3} \lor \cancel{\bar{x}_5})$, $(\cancel{\bar{x}_3} \lor \cancel{\bar{x}_5} \lor \bar{x}_8)$, $(\cancel{x_2} \lor x_6 \lor \cancel{x_8})$, $(\cancel{x_1} \lor \cancel{x_8} \lor x_9)$, $(\cancel{\bar{x}_3} \lor \cancel{\bar{x}_6} \lor \cancel{\bar{x}_9})$
$\mathrm{d}(x_1 \lor x_3)$	
$\mathrm{d}(x_1 \lor x_5)$	
$(\bar{x}_3)$	(x_1), $(\cancel{\bar{x}_1} \lor \bar{x}_2 \lor \cancel{\bar{x}_3})$, $(\cancel{\bar{x}_1} \lor \cancel{\bar{x}_3} \lor \bar{x}_4)$, $(\cancel{x_2} \lor \cancel{x_4} \lor x_6)$, $(\cancel{\bar{x}_1} \lor \cancel{\bar{x}_6} \lor \bar{x}_7)$, $(\cancel{\bar{x}_3} \lor \cancel{\bar{x}_6} \lor \bar{x}_9)$, $(\cancel{x_2} \lor \cancel{x_7} \lor \cancel{x_9})$
$(\bar{x}_5)$	(x_1), $(\bar{x}_3)$, $(\cancel{\bar{x}_1} \lor \bar{x}_4 \lor \cancel{\bar{x}_5})$, $(\cancel{\bar{x}_1} \lor \cancel{\bar{x}_5} \lor \bar{x}_6)$, $(x_2 \lor \cancel{x_4} \lor \cancel{x_6})$, $(\cancel{\bar{x}_2} \lor \cancel{\bar{x}_5} \lor \bar{x}_7)$, $(\cancel{x_3} \lor \cancel{x_4} \lor \cancel{x_7})$
$\perp$	(x_1), $(\bar{x}_3)$, $(\bar{x}_5)$, $(x_2 \lor \cancel{x_3} \lor \cancel{x_5})$, $(\cancel{x_3} \lor \cancel{x_5} \lor x_8)$, $(\cancel{\bar{x}_2} \lor \bar{x}_6 \lor \cancel{\bar{x}_8})$, $(\cancel{\bar{x}_1} \lor \cancel{\bar{x}_8} \lor \bar{x}_9)$, $(\cancel{x_3} \lor \cancel{x_6} \lor \cancel{x_9})$

are crucial for fast performance. Both disadvantages slow down the verified solver substantially, making it useless in most practical settings.

The second approach has been more successful in the context of SAT solving. We refer to a certificate of an unsatisfiability claim as a *proof of unsatisfiability*. What kind of format would be useful for such proofs? The ideal proof format facilitates five properties: (1) proof production should be *easy* to ensure that it will be supported by many solvers; (2) proofs should be *compact* in order to have small overhead; (3) proof validation should be *simple*, otherwise the trust issue persists; (4) proof validation should be *efficient* to make verification useful in practice; and (5) all techniques should be *expressible*, otherwise solvers will be handicapped. There is a trade-off between these properties. For example, more details in a proof should allow a more efficient validation procedure. However, adding details makes proofs less compact and harder to produce.

Initially, proofs of unsatisfiability were based on resolution. Although useful in some settings, it is hard or even impossible to achieve the properties of easy production (1), compactness (2), and expressibility (5) for such proofs. The alternative is *clausal proofs*[12] for which it is now possible to achieve all five properties.

What is a clausal proof of unsatisfiability for a SAT problem? Basically, we start with the given list of clauses, and add or delete clauses, until finally we add the empty clause $\bot$, which marks unsatisfiability, since there is no literal in it to satisfy. The most basic restriction on adding clauses is, that the addition is *solutions-preserving*, that is, all solutions (at that point, taking all previous additions and deletions into account) also satisfy the added clause. This guarantees correctness: if all additions are solutions-preserving, and we are able to add $\bot$ (which has no solution), then the original SAT problem must be unsatisfiable. For example, consider the formula $F = (x \vee y) \wedge (x \vee \bar{y})$. Adding the clause (x) to F is solutions-preserving: F has two solutions and in both solutions x is assigned to `true`.

It is important to validate that clause addition steps are solutions-preserving, otherwise we do not have a *proof*, just some sort of claim. This verification should be cheap to perform, and the basic criterion is as follows. Suppose a formula F is given, and the clause C is claimed to be solutions-preserving for F. Take the assignment that sets all literals in C to `false`. If UCP on F results in a conflict, then the clause is indeed solutions-preserving, since we checked that it is not possible to falsify C while satisfying F. This realizes the first three ideal proof format properties: easy, compact, and simple. The solver can just output the learned clauses, without a justification, and validation happens by UCP.

SAT solvers do not only learn lots of clauses, but also aggressively delete them to achieve fast UCP. Proofs should include this deletion information in order to realize efficient validation. Furthermore, proof checkers require dedicated UCP algorithms to make proof validation as fast as proof production.[14] Combining these techniques realizes the fourth ideal proof property (efficient validation).

A proof of our running example is shown in Figure 2. The proof consists of six clause addition steps and two clause deletion steps. The latter have a "d" prefix and do not require checking. The correctness of each clause addition step is checked using UCP, and shown using a unit clause justification: a sequence of clauses that become unit, ending with a falsified clause that marks the conflict. The unit clause justification is omitted from clausal proofs to ensure compactness, but the checker constructs a justification during validation.

Some SAT solving techniques may change (add or remove) solutions which can significantly reduce solving time. In order to express such techniques—to have also the final ideal proof property (expressible)—support is required for proof steps that go beyond the above solutions-preservation. This is realized by the concept of *solutions-preserving modulo x* for some literal x. Let φ be an assignment. We denote by $\varphi \oplus x$ the assignment obtained by flipping the truth value for literal x in φ. In case x is unassigned in φ, then x is assigned to `true` in $\varphi \oplus x$. For a given formula F, addition of clause C is solutions-preserving modulo x if for all solutions φ of F at least one of φ or $\varphi \oplus x$ satisfies F *and* C.

For example, consider the formula $F = (x \vee y) \wedge (x \vee \bar{y})$ again. The addition of clause $(\bar{x} \vee y)$ to F is solutions-preserving modulo y. Recall that F has two solutions. The first solution φ_1, where x is `true` and y is `true`, also satisfies $(\bar{x} \vee y)$. The second solution φ_2, where x is `true` and y is `false`, falsifies $(\bar{x} \vee y)$, but $\varphi_2 \oplus y$ satisfies F and $(\bar{x} \vee y)$.

How to check that adding clause C is solutions-preserving modulo x? We use the following efficient criterion: $x \in C$, and for all $D \in F$ with $\bar{x} \in D$ we have that setting all literals in C *as well as* all literals in $D \backslash \{\bar{x}\}$ to `false` yields a conflict via UCP. The proof format that encapsulates this inference in a single step is called the "DRAT" format,[4] and is supported by state-of-the-art solvers.

It is instructive to show that this criterion guarantees adding C to F is solutions-preserving modulo x. The critical clauses are the $D \in F$ with $\bar{x} \in D$, since here flipping of x might change a satisfied clause to a falsified clause. First observe that from the criterion follows that all $C \cup (D \backslash \{\bar{x}\})$ are solutions-preserving w.r.t. F. Now assume that φ is a total satisfying assignment for F which falsifies C (otherwise φ satisfies F and C and we are done). Thus φ falsifies x, and $\varphi \oplus x$ satisfies C. Since all $C \cup D \backslash \{\bar{x}\}$ are solutions-preserving w.r.t. F, φ satisfies all $C \cup D \backslash \{\bar{x}\}$. Hence φ satisfies all $D \backslash \{\bar{x}\}$ (because φ falsifies C), and so does $\varphi \oplus x$ as well, and thus indeed $\varphi \oplus x$ satisfies all D. QED

The DRAT format seems to be a good proof format for existing and future SAT solvers, as it has all the five properties of an ideal proof format. Moreover, DRAT proofs can be efficiently checked even in parallel, and they have been used to validate the results of the annual international SAT competitions since 2013. For the Boolean Schur Triples Problem with $n = 9$, there exists a DRAT proof consisting of only four clause additions: $(x_1 \vee x_4)$, (x_1), (x_4), $\bot$. Validating this proof involves more details, which can be obtained by using the DRAT proof checker DRAT-TRIM.[d]

Indeed, DRAT in a theoretical sense is equivalent to one of the most powerful systems studied in proof complexity, Extended Frege with Substitution, and thus it should offer "proofs as short as possible."[4] The Extension

[d] The tool is available at https://github.com/marijnheule/drat-trim.

Rule basically states that the clauses $(x \vee \bar{a} \vee \bar{b}) \wedge (\bar{x} \vee a) \wedge (\bar{x} \vee b)$ can be added if no literals x and $\bar{x}$ occur in the formula. In fact, each of the clauses are solutions-preserving modulo x or $\bar{x}$ according to the above criterion.

Proof size nevertheless becomes an issue. Although DRAT proofs are "compact," the size of the DRAT proof of the Boolean Pythagorean Triples Problem is 200 TB. An obvious challenge of such a huge file is its storage. Also, dealing with such files increases the complexity of proof validation algorithms, which will need to support parallel checking. On the other hand, it is possible to trade complexity for space by adding details to the proof that facilitate fast checking. In order to make this feasible, the proof can be optimized using a non-verified trimmer which also adds the checking details. This approach has been successfully applied to validate the 200 TB proof using a checker which was *formally verified* in Coq.[8]

Ramsey Theory and Complexity

A popularized summary of Ramsey Theory is that "complete chaos is impossible."[26] More concretely, Ramsey Theory deals with patterns that occur in well-known sets such as the set of natural numbers or the set of graphs. For example, coloring the natural numbers with finitely many colors will result in a monochromatic Schur triple $a + b = c$.

Hundreds of papers have been published on determining the smallest size of sets such that a given pattern must start to occur.[32] The most famous pattern is related to Ramsey numbers $R(k)$: the smallest n such that all red/blue edge colorings of the complete graph with n vertices have a red or a blue clique of size k. Only the first four Ramsey numbers are known. Paul Erdős famously told a story about aliens who threatened to obliterate earth unless humans provided them with the value of $R(5)$—with a proof, we may add here. Putting all mankind behind this project would do the job in a year. Yet if aliens asked for $R(6)$, we should opt for the Hollywood resolution and obliterate them instead.[13]

Many problems in Ramsey Theory appear to be solved only using large case splits (especially for the determination of Ramsey-type numbers), and thus using SAT is a natural option. Also SAT formulations of these problems are easy and natural. In order to determine the smallest subset in which a pattern starts to occur using SAT, two formulas need to be solved. First, it has to be shown that for any smaller subset there exists a counter-example. This is typically easy, because the formula is satisfiable. The second formula, encoding the existence of the pattern, is much harder to solve as now unsatisfiability must be shown.

The first major success of SAT solving in Ramsey Theory was determining the sixth Boolean van der Waerden number:[22] vdW(6) = 1132. The number vdW(k) expresses the smallest n such that any red/blue coloring of the numbers 1 to n results in a monochromatic arithmetic progression of length k. The computation used multiple clusters as well as dedicated SAT-solving hardware (FPGA solvers) for several months. Unfortunately, no proof was produced during the computation, making it impossible to verify the result. This raises several trust issues, because errors could have been made on several levels. For example, was the splitting correct and thus has the whole search space been explored? Also, FPGA solvers have been tested much less thoroughly compared to state-of-the-art solvers.

The first important problem with a verified clausal proof is the Erdős Discrepancy Problem (EDP), which is about "complete uniformity is impossible." The problem conjectures that any infinite sequence $s_1, s_2, \ldots$ with $s_i = \pm 1$ contains for any positive integer C a subsequence $s_d, s_{2d}, s_{3d}, \ldots, s_{kd}$, for some positive integers k and d, such that $\left|\sum_{i=1}^{k} s_{id}\right| \geq C$. Using colors, the conjecture says that for every $C \geq 1$ and every red/blue coloring of 1, 2, ... there is a finite initial segment of some progression $d, 2d, 3d, \ldots$ for some $d \geq 1$, such that the discrepancy between the number of color-occurrences is at least C (one color occurs at least C-times more than the other). The conjecture has been a long-standing open problem even for $C = 2$. The case $C = 2$ was eventually solved using SAT by providing the exact bound,[21] also applying Cube-and-Conquer. The encoding of this problem is more involved than the simple encoding of Ramsey problems (which are just hypergraph coloring problems), and thus, though a clausal proof has been provided, correctness is more of an issue than in cases of Ramsey Theory. Computationally, EDP is much easier,[21] and a much smaller proof exists (about a gigabyte) than in our case. Finally a general mathematical existence proof has been provided.[35] This mathematical proof was called "much more satisfying" than the computational approach.[25] However, there is for example the possibility that the Pythagorean Tuples Conjecture (see below) is not provable with current methods. Furthermore, the SAT approach is actually a rather "satisfying approach" when taking into account its deep connections to formal methods.

The *Pythagorean Tuples Conjecture* states that Ptn(k; m)—with k the length of the tuple and m the number of colors—exists for all $k \geq 3$ and $m \geq 2$. That is, for every partitioning of $\{1, \ldots, \mathrm{Ptn}(k; m)\}$ into m parts, some part contains a Pythagorean tuple of size k. We have shown that Ptn(3; 2) = 7825. The value of Ptn(3; 2) was conjectured[30] not to exist after determining the numbers Ptn(k; 2) for $4 \leq k \leq 31$. We have meanwhile computed the only known Pythagorean tuples numbers for three colors: Ptn(5; 3) = 191, Ptn(6; 3) = 121, and Ptn(7; 3) = 102. We also established Ptn(3; 3) > 10^7, and this lower bound (via local-search algorithms) seems still far away from the exact bound. So it is imaginable that a mathematical existence-proof can not be found, and finiteness of Ptn(3; 3) might never be established. It is furthermore conceivable that the Pythagorean Tuples Conjecture is true but the best proofs are SAT-like. Thus formal proofs in systems like Zermelo-Fraenkel set theory would only *exist* for concrete k and m, while there would not exist a single proof for all k and m. No mathematical existence proofs have yet been established for any Ptn(k; m) (see "alien truth statements" for further discussions).

Before coming to the industrial applications of SAT, we remark that the Ramsey numbers[33] $R(k)$ are very different from the Boolean Pythagorean Triples Problem: namely the latter is "random-like" and thus has no symmetries (besides the trivial color symmetries). Currently SAT solving is more successful in the absence of strong symmetries, while Ramsey numbers currently have too much structure for an automated attack. More sophisticated symmetry-breaking techniques are required to improve the performance.

Brute Force Formal Methods

SAT solvers are a key technology in formal methods for applications, such as bounded model checking[5] and equivalence checking. In bounded model checking, given a transition system and an invariant such as a safety property, the SAT solver determines for some appropriate finitization, whether there exists a sequence of transitions that violates the safety property. Equivalence checking is used to determine the equivalence of a specification and an implementation or two different implementations. The SAT solver is asked to find an input such that some output differs. Notice that the existence of a solution means that the safety property is violated or that there exists a counter-example for equivalence.

All problems discussed so far could be expressed as a propositional formula. For many interesting problems, however, this is not the case and they require a richer logic for its representation. That does not mean that SAT technology cannot be used to solve these problems. On the contrary: more and more problems that require a richer logic are being solved efficiently using SAT.

The key idea is to abstract away those parts of a given problem that cannot be expressed as propositional logic. A solution of the abstracted problem may not be a solution of the given problem, while a refutation of the abstracted problem is also a refutation of the given problem. In case a solution of the abstracted problem is obtained, which is not a solution for the given problem, then the abstraction is refined by adding a clause that prevents the SAT solver from finding that solution (and potentially similar solutions) again. This is repeated until either a refutation or a solution for the given problem is found. Incremental SAT solving[10] facilitates an efficient implementation of this approach.

This approach has been very successful in Automated Theorem Proving (ATP). The long-time champion in the annual ATP competitions is Vampire,[36] which has been tightly integrated with a SAT solver. Other strong ATP solvers, including iProver and Leo, incorporate SAT solvers as well. The major interactive theorem provers, such as ACL2, Coq, and Isabelle, support the usage of SAT solvers to deal with subproblems that can be expressed in propositional logic. In this setting, SAT solvers are treated as a black-box and the emitted proofs are validated in the theorem provers. Another successful extension of SAT in this direction is *Satisfiability Modulo Theories* (SMT).[9] It uses multiple theories, such as linear arithmetic, uninterpreted functions, and bit-vectors, and replaces constraints in a theory by propositional variables. SMT solvers, such as Z3, Boolector, CVC4, and Yices have been highly successful.

More and more problems that require a richer logic are being solved efficiently using SAT.

Alien Truths

The core argument against solving a problem by brute force is it does not contribute to understanding the problem. In that view, the proof is meaningless and hard to generalize, and a human mathematical proof is preferred. Furthermore, without understanding errors seem more likely, although validation can be done by highly trusted systems.

The proponents of "elegant" proofs appear to consider problems with only very long proofs as not interesting or not relevant. But even unprovable statements, like the famous Continuum Hypothesis, have an important place in mathematics. If we do not study the limits of our current knowledge, we will stay ignorant forever, always restricted to a "safe space," neglecting problems we *assume* to be too hard. Furthermore, what is a limit of one discipline is a core subject of another discipline. Computational complexity and Ramsey Theory have close relations. *Understanding* the hardness of problems from Ramsey instances could lead to major breakthroughs.[27] For example, *why* is proving the Ramsey property for $a + b = c$ rather easy, while $a^2 + b^2 = c^2$ appears to be a very hard problem? In general, even small propositional problems might have only very large proofs. If we would ignore this area, then we would allow random holes in our knowledge. The question "why there are *no* short proofs," and "what makes a problem hard," are deep and fascinating questions, and we consider them some of the most important problems of our times.

To better discuss the untold stories of computer science, complexity theory, and SAT, let's call *alien* a provable and rather short mathematical statement with only a very long proof. Artificial alien statements can be constructed using Gödel's methods. Whether a natural truth statement can be shown to be alien, such as the Pythagorean Triples Problem, is of highest relevance. Even if a short proof for the Pythagorean Triples Problem may be constructed, that is unlikely to be the case for the exact bound result. Now there is actually a whole spectrum of possibilities between human truths and alien truths. Classical mathematical statements for which a paper proof exists, such as Schur's Theorem,[34] we consider as *human* truth statements. Hence the vast body of mathematical works belongs to this category. Furthermore, we consider mathematical statements that have been proven mostly manually, but with some computer help, *weakly human*. More specifically, such statements have a large case-split, which could potentially be understood by humans, but which have only been checked mechanically. An example of such a statement is the Four-Color Theorem.[37] The proof by Appel and Haken considers 663 cases in its improved version. The case-split is fully understood and humanly constructed. A theorem prover only checks the cases. Coming to larger cases, we refer to a *weakly alien* truth statement as a giant humanly generated case-split which can be validated using plain brute-force methods. For example, it has been shown that the minimum number of givens is 17 in Sudoku by enumerating all possible cases with 16 givens and refuting them all[29] (5 472 730 538 cases after symmetry breaking). Although impossible to evaluate by humans, it could be directly done mechanically. This result is expected to be weakly alien, as it is unlikely that there exists a small enough case-split that is checkable by humans.

We arrive at a better understanding of "alien," namely a truth statement is *alien* if humanly understandable case-splits are way too big for any plain brute-force method, but there exists a giant case-split that mysteriously avoids an enormous exponential effort. Examples of truth statements that are expected to be alien are that vdW(6) = 1132 (see Kouril[22]) and that the exact bound of EDP with $C = 2$ is 1161 (see Konev[21]). A plain brute-force approach to those problems would require the evaluation of 2^{1132} and 2^{1161} cases, respectively. Brute reasoning using SAT solvers significantly reduced the size of the case-splits and allowed determining their truth. We think it is relevant to make a further distinction: the above two alien truth statements express the exact bound, but for both cases there is a mathematical existence proof that the pattern cannot be avoided indefinitely. Now also high-level statements, such as any red/blue coloring of the natural numbers yields a monochromatic Pythagorean triple, could be alien, when the bound result, Ptn(3; 2) = 7825, is the only proof. We call such statements indeed *strongly alien*. If a mathematical existence proof would be found for the statement here, then only the bound statement remains, which is simply *alien*. This happened for the Erdős Discrepancy Problem: the bound was computed using SAT, and later a mathematical existence proof was given.

For some truth statements, we may never be able to produce a proof.

Finally, for some truth statements, we may never be able to produce a proof. A possible example problem of this type is the statement that every 3-coloring of the natural numbers yields a monochromatic Pythagorean triple. As already discussed, experiments show that Ptn(3; 3) > 10^7, where lower bounds are relatively easy to compute. Proofs of upper bound results are much harder to obtain: for example, Ptn(3; 2) > 7824 can be computed in one CPU-minute with local search, while computing Ptn(3; 2) ≤ 7825 required more than 40 000 CPU-hours. We call decidable truth statements *extra alien* if a proof can never be computed.

The concept of alien truth statements deals with the *size* of proofs, but it touches naturally on *unprovability* (in current systems like Zermelo-Fraenkel set theory). It is conceivable that Ptn(3; 3) does not exist, that is, the natural numbers are 3-colorable without a monochromatic Pythagorean triple. However, this may not be provable, since the coloring is too complex. On the other hand, it is conceivable that all Ptn(3; m) with $m \geq 3$ exist (note that a SAT solver can prove them in principle), but these statements are all alien or extra-alien. Since these proofs grow with m, the general statement that all Ptn(3; m) with $m \geq 3$ exist, is then unprovable *in principle*.

Conclusion

Recent successes in brute reasoning, such as solving the Erdős Discrepency Problem and the Pythagorean Triples Problem, show the potential of this approach to deal with long-standing open mathematical problems. Moreover, proofs for these problems can be produced and verified completely automatically. These proofs may be big, but we argued that compact elegant proofs may not exist for some of these problems, in particular (but not only) for the exact bound results. The size of these proofs does not influence the level of correctness, and these proofs may reveal interesting information about the problem.

In contrast to popular belief, mechanically produced huge proofs can actually help in understanding the given problem. We can try to understand their structure, and making them thus smaller. Hardly any research has been done yet in this direction apart from removing redundancy in a given proof. Possibilities are changing the heuristics of a solver or introducing new definitions of frequently occurring patterns in the proof. Indeed, simply validating a clausal proof does not only produce a yes/no answer as to whether the proof is correct, but also provides an *unsatisfiable core* consisting of all original clauses that were used to validate the proof—revealing important parts of the problem. The size of the core depends on the type of problem. Problems in Ramsey Theory typically have quite a large core and therefore provide limited insight. Many bounded model checking problems, however, have small unsatisfiable cores, thereby showing that large parts of the hardware design were not required to determine the safety property.

To conclude, it is definitely possible to gain insights by using SAT. However that "insight" might need to be reinterpreted here, and might work on a higher level of abstraction. Every paradigm change means asking different questions. Gödel's Incompleteness Theorem *solved* partially the question of the consistency of mathematics by showing that the answer provably cannot be delivered in the näive way. Now the task is to live up to big complexities, and to embrace the new possibilities. Proofs must become objects for investigations, and understanding will be raised to the next level, how to find and handle them.

So, when the day finally comes and the aliens arrive and ask us about Ptn(3; 3), we will tell them: "You know what? Finding the answer yourself gives you a much deeper understanding than just telling you the answer—here you have the SAT solving methodology, that's the real stuff!" And then humans and aliens will live happily ever after.

> Wir müssen wissen. Wir werden wissen.
> (We must know. We will know.)
> David Hilbert, 1930

References

1. Ahmed, T., Kullmann, O., Snevily, H. On the van der Waerden numbers w(2; 3, *t*). *Disc. Appl. Math. 174* (2014), 27–51.
2. Biere, A., Fröhlich, A. Evaluating CDCL variable scoring schemes. In *SAT* (Springer, 2015), 405–422.
3. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. eds. *Handbook of Satisfiability*, volume 185 of *FAIA*. IOS Press, Amsterdam, The Netherlands, Feb. 2009.
4. Buss, S. Propositional proofs in Frege and Extended Frege systems (abstract). In *Computer Science—Theory and Applications* (Springer, 2015), 1–6.
5. Clarke, E.M., Biere, A., Raimi, R., Zhu, Y. Bounded model checking using satisfiability solving. *Formal Methods in System Design 19*, 1 (2001), 7–34.
6. Cook, S.A. The complexity of theorem-proving procedures. In *STOC* (1971), 151–158.
7. Copty, F., Fix, L., Fraer, R., Giunchiglia, E., Kamhi, G., Tacchella, A., Vardi, M.Y. Benefits of bounded model checking at an industrial setting. In *CAV* (Springer, 2001), 436–453.
8. Cruz-Filipe, L., Marques-Silva, J.P., Schneider-Kamp, P. Efficient certified resolution proof checking, 2016. https://arxiv.org/abs/1610.06984.
9. de Moura, L., Bjørner, N. Satisfiability modulo theories: Introduction and applications. *Communications of the ACM 54*, 9 (2011), 69–77.
10. Eén, N., Sörensson, N. Temporal induction by incremental SAT solving. *Electr. Notes Theor. Comput. Sci. 89*, 4 (2003), 543–560.
11. Garey, M.R., Johnson, D.S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
12. Goldberg, E.I., Novikov, Y. Verification of proofs of unsatisfiability for CNF formulas. In *DATE* (IEEE, 2003), 10886–10891.
13. Graham, R.L., Spencer, J.H. Ramsey theory. *Scientific American 263*, 1 (July 1990), 112–117.
14. Heule, M.J.H., Hunt, W.A. Jr., Wetzler, N. Trimming while checking clausal proofs. In *FMCAD* (IEEE, 2013), 181–188.
15. Heule, M.J.H., Kullmann, O., Marek, V.W. Solving and verifying the Boolean Pythagorean Triples problem via Cube-and-Conquer. In *SAT* (Springer, 2016) 228–245.
16. Heule, M.J.H., Kullmann, O., Wieringa, S., Biere, A. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In *HVC* (Springer, 2011), 50–65.
17. Heule, M.J.H., van Maaren, H. Look-ahead based SAT solvers. In Biere et al. [3], Chapter 5, (2009), 155–184.
18. Ivančić, F., Yang, Z., Ganai, M.K., Gupta, A., Ashar, P. Efficient SAT-based bounded model checking for software verification. *Theoretical Computer Science 404*, 3 (2008), 256–274.
19. Karp, R.M. Reducibility among combinatorial problems. In *Complexity of Computer Computations* (Plenum Press, 1972), 85–103.
20. Kautz, H.A., Sabharwal, A., Selman, B. Incomplete algorithms. In Biere et al. [3], Chapter 6, (2009), 185–203.
21. Konev, B., Lisitsa, A. Computer-aided proof of Erdös discrepancy properties. *Artificial Intelligence 224*, C (July 2015), 103–118.
22. Kouril, M., Paul, J.L. The van der Waerden number *W*(2, 6) is 1132. *Experimental Mathematics 17*, 1 (2008), 53–61.
23. Kullmann, O. Fundaments of branching heuristics. In Biere et al. [3], Chapter 7, (2009), 205–244.
24. Lam, C.W.H. The search for a finite projective plane of order 10. *The American Mathematical Monthly 98*, 4 (April 1991), 305–318.
25. Lamb, E. Maths proof smashes size record: Supercomputer produces a 200-terabyte proof—but is it really mathematics? *Nature 534* (June 2016), 17–18.
26. Landman, B.M., Robertson, A. *Ramsey Theory on the Integers*, volume 24 of *Student mathematical library*. American Mathematical Society, Providence, RI, 2003.
27. Lauria, M., Pudlák, P., Rödl, V., Thapen, N. The complexity of proving that a graph is Ramsey. In *ICALP* (Springer, 2013), 684–695.
28. Marques-Silva, J.P., Lynce, I., Malik, S. Conflict-driven clause learning SAT solvers. In Biere et al. [3], Chapter 4, (2009), 131–153.
29. McGuire, G., Tugemann, B., Civario, G. There is no 16-clue Sudoku: Solving the Sudoku minimum number of clues problem via hitting set enumeration. *Experimental Mathematics 23*, 2 (2014), 190–217.
30. Myers, K.J. *Computational advances in Rado numbers*. PhD thesis, Rutgers University, New Brunswick, NJ, 2015.
31. Oe, D., Stump, A., Oliver, C., Clancy, K. versat: A verified modern SAT solver. In *VMCAI* (Springer, 2012) 363–378.
32. Radziszowski, S.P. Small Ramsey numbers. *The Electronic Journal of Combinatorics* (January 2014), Dynamic Surveys DS1, Revision 14.
33. Ramsey, F.P. On a problem of formal logic. *Proceedings of the London Mathematical Society 30* (1930), 264–286.
34. Schur, I. Über die Kongruenz $x^m + y^m = z^m$ (mod *p*). *Jahresbericht der Deutschen Mathematiker-Vereinigung 25* (1917), 114–116.
35. Tao, T. The Erdös discrepancy problem. *Discrete Analysis 1* (February 2016), 29.
36. Voronkov, A. AVATAR: The architecture for first-order theorem provers. In *CAV* (Springer, 2014) 696–710.
37. Wilson, R. *Four Colors Suffice: How the Map Problem Was Solved*. Princeton University Press, Princeton, NJ, revised edition, 2013.

Marijn J.H. Heule (marijn@cs.utexas.edu) is a research scientist at The University of Texas, Austin.

Oliver Kullmann (o.kullmann@swansea.ac.uk) is an associate professor in computer science at Swansea University, U.K.

Watch the authors discuss their work in this exclusive *Communications* video. https://cacm.acm.org/videos/the-science-of-brute-force

research highlights

P. 82
Technical Perspective
Unexpected Connections
By Marc Shapiro

P. 83
The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors
By Austin T. Clements, M. Frans Kaashoek, Eddie Kohler, Robert T. Morris, and Nickolai Zeldovich

P. 91
Technical Perspective
Linking Form, Function, and Fabrication
By Helmut Pottmann

P. 92
Spin-It: Optimizing Moment of Inertia for Spinnable Objects
By Moritz Bächer, Bernd Bickel, Emily Whiting, and Olga Sorkine-Hornung

DOI:10.1145/3068768

To view the accompanying paper, visit doi.acm.org/10.1145/3068914 rh

Technical Perspective
Unexpected Connections

By Marc Shapiro

SCALABILITY IS THE capability of a parallel program to speed up its execution as we provide it with more CPUs. Back in 1967, Gene Amdahl noticed the sequential part of a parallel program had a disproportionate influence on scalability.[1] Suppose that some program takes 100 s to run on a sequential processor. Now, let's run it on a parallel computer. If we are able to parallelize, say, 80% of the code, then with enough CPUs that 80% would take essentially zero time. However, the remaining sequential portion will not run any faster; this means the parallel program will always take at least 20 s to run, a maximum speed-up of only 5×. If we are able to parallelize 95% of the code, speedup is still limited to 20×, even with an infinite number of CPUs! This back-of-the-envelope calculation, known as Amdahl's Law, does not take into account other factors, such as increased memory size, but remains an important guideline.

In 1967, parallelism was a niche topic, but not any more. Improving program performance on today's clusters, clouds, and multicore computers requires the developer to pay serious attention to scalability. The inherent scalability of an interface is the focus of the following paper.

When a thread updates some shared datum, and another thread wants to read or write the most recent version of that datum (they *conflict*), they must synchronize, which constitutes a sequential bottleneck. This is a general result that does not depend on any particular implementation, even with efficient hardware support for cache coherence, as explained by the authors.

Here comes the paper's main insight: If two concurrent procedure calls *commute* with each other (that is, executing them in either order is equivalent), this means that neither one depends on the result of the other. Therefore, *there is no inherent reason why these calls should conflict; and, hence, it is possible to implement them in a way that scales well.*

> **The following paper presents a simple and powerful idea. It is not just about OSs, but applies to any piece of parallel software, whether running on a multicore computer or in the cloud.**

The advantages of commutativity in software have been known for a long time, see the paper for relevant references. It is only recently, however, that focus has shifted from simply leveraging existing commutativity toward designing software to achieve commutativity.[2,3] The paper goes well beyond previous work. First, instead of simple abstract data types, it considers the more complex case of software with an intricate interface and massive amount of shared state—a whole operating system (OS). Second, instead of just a black-and-white characterization "commute/don't-commute," it considers calls that may commute in some states and not in others. This is especially important when commuting is the common case, as in many OS calls. Finally, it leverages static program verification techniques, providing a tool that will prove if and when a given interface is commutative, and will generate test cases exercising the scalability of its implementation.

The authors designed a whole OS based on these ideas. It's similar to Linux, but its APIs are designed for commutativity. The implementation is mostly scalable, but not always: even when a scalable implementation of an API exists in theory, it will not necessarily be the most obvious or even the most efficient; sometimes, it's simply not worthwhile. They also learned that many advanced data structures do not scale well; for instance, rebalancing a tree might modify a portion of the tree that is semantically unrelated to the update that triggered the rebalancing.

The authors present a simple and powerful idea. It is not just about OSs, but applies to any piece of parallel software, whether running on a multicore computer or in the cloud. Commutativity enables us to reason about scalability in a principled way, independently of a particular implementation, benchmark or workload. We can now design our APIs to be scalable, by ensuring calls commute in the common case, and we can use verification tools to automate and exercise this reasoning—an unexpected connection between high-school math theory and hardcore computer science. C

References

1. Amdahl, G.M. Validity of the single-processor approach to achieving large scale computing capabilities. In *Proceedings of the AFIPS Conference, 30* (Atlantic City, NJ, Apr. 1967) AFIPS Press.
2. Shapiro, M., Preguiçą, N., Baquero, C. and Zawirski, M. Conflict-free replicated data types. In *Proceedings of the Int. Symp. on Stabilization, Safety, and Security of Dist. Sys. 6976.* Lecture Notes in Comp. Sc. X. Défago, F. Petit, and V. Villain, Eds. Grenoble, France, Oct. 2011, 386–400,. Springer-Verlag; doi: 10.1007/978-3-642-24550-3 29; http://www.springerlink.com/content/3rg39l2287330370/.
3. Shapiro, M., Preguiçą, N., Baquero, C. and Zawirski, M. Convergent and commutative replicated data types. *Bulletin of the European Association for Theoretical Computer Science 104* (June 2011), 67–88; http://www.eatcs.org/images/bulletin/beatcs104.pdf.

Marc Shapiro is a Principal Researcher in the Regal group of UPMC-LIP6 and Inria, Paris, France.

DOI:10.1145/3068914

The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors

By Austin T. Clements, M. Frans Kaashoek, Eddie Kohler, Robert T. Morris, and Nickolai Zeldovich

Abstract

Developing software that scales on multicore processors is an inexact science dominated by guesswork, measurement, and expensive cycles of redesign and reimplementation. Current approaches are workload-driven and, hence, can reveal scalability bottlenecks only for known workloads and available software and hardware. This paper introduces an *interface-driven* approach to building scalable software. This approach is based on the *scalable commutativity rule*, which, informally stated, says that whenever interface operations commute, they can be implemented in a way that scales. We formalize this rule and prove it correct for any machine on which conflict-free operations scale, such as current cache-coherent multicore machines. The rule also enables a better design process for scalable software: programmers can now reason about scalability from the earliest stages of interface definition through software design, implementation, and evaluation.

1. INTRODUCTION

Until the mid-2000s, continuously rising CPU clock speeds made sequential software perform faster with each new hardware generation. But higher clock speeds require more power and generate more heat, and around 2005 clock speeds reached the thermal dissipation limits of a few square centimeters of silicon. CPU architects have not significantly increased clock speeds since, but the number of transistors that can be placed on a chip has continued to rise. Architects now increase parallelism by putting more CPU cores on each chip. *Total* cycles per second continues to grow exponentially, but software must *scale*—must take advantage of parallel CPU resources—to benefit from this growth.

Unfortunately, scaling is still an untamed problem. Even with careful engineering, software rarely achieves the holy grail of linear scalability, where doubling hardware parallelism doubles software performance.

Engineering scalable systems software is particularly challenging. Systems software, such as operating system kernels and databases, presents services to applications through well-defined interfaces. Designers rarely know ahead of time how applications will use these interfaces, and thus often cannot predict what bottlenecks to multicore scalability will arise. Furthermore, scaling bottlenecks may be a consequence of the definition of the interface itself; such problems are particularly difficult to address once many applications depend on the interface.

Lack of a principled way to reason about scalability hampers all phases of systems software development: defining an interface, implementing the interface, and testing its scalability.

When defining an interface, developers lack a systematic way of deciding whether a given definition will allow for scalable implementations. Demonstrating a scalability bottleneck requires a complete implementation and a workload. By the time these are available, interface changes may no longer be practical: many applications may rely on the existing interface, and applications that trigger the bottleneck may not be important enough to warrant an interface change.

During design and implementation, developers lack a systematic way to spot situations in which perfect scalability is achievable. This makes it hard to design an implementation to be scalable from the start. Instead, over time developers must iteratively improve the software's parallel performance as specific workloads uncover bottlenecks, often re-implementing the software multiple times.

While testing, developers lack a systematic way of evaluating scalability. The state of the art for testing the scalability of multicore software is to choose a workload, plot performance at varying numbers of cores, and use tools such as differential profiling[13] to identify scalability bottlenecks exhibited by that workload. Each new hardware model or workload, however, may expose new scalability bottlenecks.

This paper presents a new approach to designing scalable software that starts with the design of scalable software interfaces. This approach makes it possible to reason about multicore scalability before an implementation exists, and even before the necessary hardware is available. It can highlight inherent scalability problems, leading to better interface designs. It sets a clear scaling target for the implementation of a scalable interface. Finally, it enables systematic testing of an implementation's scalability.

At the core of our approach is what we call the *scalable commutativity rule*: In any situation where several operations *commute* (meaning there is no way to distinguish their execution order using the interface), there exists an implementation that is *conflict-free* during those operations (meaning no core writes a cache line that was read or written

The original version of this paper was published in the *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP'13).*

by another core). Since conflict-free operations empirically scale (as we argue in Section 2), this implementation scales. Thus, more concisely, *whenever interface operations commute, they can be implemented in a way that scales*.

This rule makes intuitive sense: when operations commute, their results (return values and effect on system state) are independent of order. Hence, communication between commutative operations is unnecessary and avoiding it yields a conflict-free implementation. Conflict-free operations can execute on different cores without mutual interference via inter-core cache coherence invalidation requests, allowing total throughput to scale linearly with the number of cores.

The intuitive version of the rule is useful in practice, but is not precise enough to reason formally about interfaces or to build automated tools that evaluate scalability. This paper formalizes the scalable commutativity rule and illustrates its usefulness in the context of several examples, and for entire operating systems that support POSIX, a complicated, widely used interface.

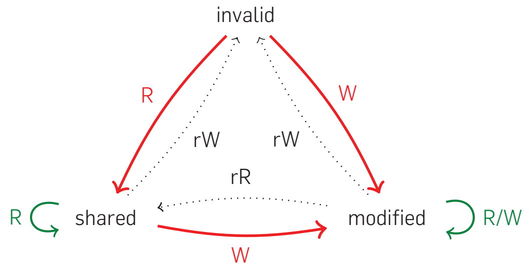
2. SCALABILITY AND CONFLICT-FREEDOM

The scalable commutativity rule assumes that code with conflict-free memory accesses—that is, code in which no cache line written by one core is read or written by any other core—is scalable. This section argues that, under reasonable assumptions, conflict-free operations do scale linearly on shared-memory multicore computers.

Multicores maintain a unified, globally consistent view of memory using MESI-like coherence protocols.[15] MESI protocols coordinate ownership of cached memory at the level of cache lines. Their key invariant is that a line with a mutable copy in one core's cache cannot be present in any other caches: obtaining a mutable copy invalidates any other caches' immutable copies. This requires coordination, which affects scalability.

Figure 1 shows the basic state machine implemented by each cache for each cache line. This maintains the invariant by ensuring a cache line is either "invalid" in all caches, "modified" in one cache and "invalid" in all others, or "shared" in any number of caches. Practical implementations add further states—MESI's "exclusive" state, Intel's "forward" state, and AMD's "owned" state—but these do not change the basic communication required to maintain cache coherence.

Figure 1. A basic cache-coherence state machine. "R" and "W" indicate local read and write operations, while "rR" and "rW" indicate reactions to remote read and write operations. Thick red lines show operations that cause communication. Thin green lines show operations that occur without communication.



Roughly, a set of operations scales when maintaining coherence does not require ongoing communication. There are two memory access patterns that fit:

- Multiple cores reading and/or writing different cache lines. This scales because no further communication is required once each cache line is in the relevant core's cache, so further accesses can proceed independently of concurrent operations.
- Multiple cores reading the same cache line. A copy of the line can be kept in each core's cache in shared mode; further reads from those cores can access the line without communication.

That is, when memory accesses are conflict-free, they do not require communication. Furthermore, higher-level operations composed of conflict-free reads and writes are themselves conflict-free and will also execute independently and in parallel. In all of these cases, conflict-free operations execute in the same time in isolation as they do concurrently, so the total throughput of N such concurrent operations is proportional to N. Therefore, given a perfect implementation of MESI, conflict-free operations scale linearly.

Conflict-freedom is not a perfect predictor of scalability. Limited cache capacity and associativity cause caches to evict cache lines (later resulting in cache misses) even in the absence of coherence traffic, and a core's first access to a cache line will always miss. Such misses directly affect sequential performance, but they may also affect the scalability of conflict-free operations. Satisfying a cache miss (due to conflicts or capacity) requires the cache to fetch the cache line from another cache or from memory; the resulting communication may contend with concurrent operations for interconnect resources or memory controller bandwidth. But applications with good cache behavior are unlikely to have such problems, while applications with poor cache behavior usually have sequential performance problems that outweigh scalability concerns. We have verified on real hardware that conflict-free operations actually do scale linearly under reasonable workload assumptions.[6]

3. THE SCALABLE COMMUTATIVITY RULE

Connections between commutativity and scalability have been explored before, especially in the context of operations on abstract data types.[2, 16, 17, 19, 21, 22] For instance, commutative replicated data types[19] are distributed objects whose operations always commute, allowing scalable, synchronization-free implementation. Abstract data type operations commute if they always produce the same result, regardless of order. For example, set member insertion commutes with itself, but not with removal: set.insert(i) and set.insert(j) produce the same results in either order, set.insert(i) and set.remove(j) has order-dependent results if i = j. But the systems interfaces we care about are richer, more granular, and more state- and context-dependent than typical data type operations. Consider the POSIX creat system call, which creates a file. The calls creat("/d1/x") and creat("/d2/y") seem to commute: their results are the same, regardless of the order they are applied. But if the disk is almost full and only one inode remains, then the calls do not commute—the second creat call will fail. (Unless, that is, one or more of the files already exists, in which case the calls commute after all!) Special cases like this can dominate analyses that use a strong notion of commutativity. If commutative operations had to commute in *all* contexts, then only trivial systems operations could commute, and commutativity would not help us explore interface scalability.

Our work relies on a new definition of commutativity, called *SIM commutativity* (State-dependent, Interface-based, and Monotonic), that captures state- and context-dependence, and conditional commutativity, independent of any implementation. SIM commutativity lets us prove the scalable commutativity rule, which says that scalable implementations exist whenever operations commute. Even if an interface is commutative only in a restricted context, there exists an implementation that scales in that context.

The rest of this section explains this formalism, gives the rule precisely, and lays out some of its consequences for system designers.

3.1. Specifications

We represent specifications using *actions*, where an action is either an *invocation* (representing an operation call with arguments) or a *response* (representing the return value). Splitting each operation into an invocation and a response lets us model blocking interfaces and concurrent operations.[11] Each invocation is made by a specific thread and the corresponding response is returned to the same thread. We will write invocations as creat(“/x”)_1 and responses as $\overline{\text{OK}}_1$, where an overbar marks responses and subscript numbers are thread IDs.

A particular execution of a system is a *history* or *trace*, which is just a sequence of actions. For example,

$$H = [\mathrm{A}_1, \mathrm{B}_3, \mathrm{C}_2, \overline{\mathrm{A}}_1, \overline{\mathrm{C}}_2, \overline{\mathrm{B}}_3, \mathrm{D}_1, \overline{\mathrm{D}}_1, \mathrm{E}_2, \mathrm{F}_3, \mathrm{G}_1, \overline{\mathrm{E}}_2, \overline{\mathrm{G}}_1, \overline{\mathrm{F}}_3],$$

consists of seven invocations and seven corresponding responses across three different threads. In a *well-formed* history, each thread's actions alternate invocations and responses, so each thread has at most one outstanding invocation at any point. H above is well-formed; for instance, in the thread-restricted subhistory $H|1 = [\mathrm{A}_1, \overline{\mathrm{A}}_1, \mathrm{D}_1, \overline{\mathrm{D}}_1, \mathrm{G}_1, \overline{\mathrm{G}}_1]$, which selects 1's actions from H, invocations and responses alternate as expected.

A *specification* models an interface's behavior as a set of system histories—specifically, a prefix-closed set of well-formed histories. A system execution is "correct" according to the specification if its trace is included in the specification. For instance, if $\mathscr{S}$ corresponded to the POSIX specification, then $[\text{getpid}_1, \overline{92}_1] \in \mathscr{S}$ (a process may have PID 92) but $[\text{getpid}_1, \overline{\text{ENOENT}}_1] \notin \mathscr{S}$ (the getpid() system call may not return that error). A specification constrains both invocations and responses: $[\text{NtAddAtom}_1]$ is not in the POSIX specification because NtAddAtom is not a POSIX system call.

An *implementation* is an abstract machine that takes invocations and calculates responses. Our constructive proof of the scalable commutativity rule uses a class of machines on which conflict-freedom is defined[6]; a good analogy is a Turing-type machine with a random-access tape, where conflict-freedom follows if the machine's operations on behalf of different threads access disjoint portions of the tape. An implementation may "stutter-step," taking multiple rounds to finish calculating a response, and it does not have to generate responses in the order invocations were received.

An implementation M *exhibits* a history H if, when fed H's invocations at the appropriate times, M can produce H's responses (so that its external behavior equals H overall). An implementation M is *correct* for a specification $\mathscr{S}$ if M's responses always obey the specification. This means that every history exhibited by M is either in $\mathscr{S}$, or contains some invalid invocation.

3.2. Commutativity

SIM commutativity, which we define here, aims to capture state dependence at the interface level. State dependence means SIM commutativity must capture when operations commute in some states, even if those same operations do not commute in other states; however, we wish to capture this contextually, without reference to any particular implementation's state. To reason about *possible* implementations, we must capture the scalability inherent in the interface itself. This in turn makes it possible to use the scalable commutativity rule early in software development, during interface design and initial implementation.

Commutativity states that actions may be reordered without affecting eventual results. We say a history H' is a *reordering* of H when $H|t = H'|t$ for every thread t. This allows actions to be reordered across threads, but not within them. For example, if $H = [\mathrm{A}_1, \mathrm{B}_2, \overline{\mathrm{A}}_1, \mathrm{C}_1, \overline{\mathrm{B}}_2, \overline{\mathrm{C}}_1]$, then $[\mathrm{B}_2, \overline{\mathrm{B}}_2, \mathrm{A}_1, \overline{\mathrm{A}}_1, \mathrm{C}_1, \overline{\mathrm{C}}_1]$ is a reordering of H, but $[\mathrm{B}_2, \mathrm{C}_1, \overline{\mathrm{B}}_2, \overline{\mathrm{C}}_1, \mathrm{A}_1, \overline{\mathrm{A}}_1]$ is not, since it does not respect the order of actions in $H|1$.

Now, consider a history $H = X \parallel Y$ (where $\parallel$ concatenates action sequences).We say Y *SI-commutes* in H when given any reordering Y' of Y, and any action sequence Z,

$$X \parallel Y \parallel Z \in \mathscr{S} \quad \text{if and only if} \quad X \parallel Y' \parallel Z \in \mathscr{S}.$$

This definition captures state dependence at the interface level. The action sequence X puts the system into a specific state, without specifying a representation of that state (which would depend on an implementation). Switching regions Y and Y' requires that the exact responses in Y remain valid according to the specification even if Y is reordered. The presence of region Z in both histories requires that reorderings of actions in region Y cannot be distinguished by future operations, which is an interface-based way of saying that Y and Y' leave the system in the same state.

Unfortunately, SI commutativity is not sufficient to prove the scalable commutativity rule. To avoid certain degenerate cases, we must further strengthen the definition of commutativity to be *monotonic* (the M in SIM). An action sequence Y *SIM-commutes* in a history $H = X \parallel Y$ when for any *prefix* P of any reordering of Y (including $P = Y$), P SI-commutes in $X \parallel P$. Equivalently, Y SIM-commutes in H when, given any prefix P of any reordering of Y, any reordering P' of P, and any action sequence Z,

$$X \parallel P \parallel Z \in \mathscr{S} \quad \text{if and only if} \quad X \parallel P' \parallel Z \in \mathscr{S}.$$

Both SI commutativity and SIM commutativity capture state dependence and interface basis. Unlike SI commutativity, SIM commutativity excludes cases where the commutativity of a region changes depending on future operations. SIM commutativity is what we need to state and prove the scalable commutativity rule.

3.3. Rule

We can now formally state the scalable commutativity rule.

> Assume an interface specification $\mathscr{S}$ that has a correct implementation and a history $H = X \parallel Y$ exhibited by that implementation. Whenever Y SIM-commutes in H, there exists a correct implementation of $\mathscr{S}$ whose steps in Y are conflict-free. Since, given reasonable workload assumptions, conflict-free operations empirically scale on modern multicore hardware, this implementation is scalable in Y.

Our proof of the rule constructs the scalable implementation from the correct reference implementation, and relies on our abstract machine definition and our definition of conflict-freedom.[6]

3.4. Example

Consider a reference counter interface with four operations. reset(v) sets the counter to a specific value v, inc and dec increment and decrement the counter and return its new value, and isz returns Z if the counter value is zero and NZ otherwise. The caller is expected to never decrement below zero, and once the counter reaches zero, the caller should not invoke inc.

Consider the counter history

$$H = [\mathsf{reset}(2)_3, \overline{\mathsf{OK}}_3, \underbrace{\mathsf{isz}_1, \overline{\mathsf{NZ}}_1, \mathsf{isz}_2, \overline{\mathsf{NZ}}_2}_{H_1}, \underbrace{\mathsf{dec}_3, \overline{1}_3, \mathsf{dec}_4, \overline{0}_4}_{H_2}].$$

The region H_1 SIM-commutes in H, so the rule tells us that a correct implementation exists that is conflict-free for H_1. In fact, this is already true of a simple shared-counter implementation: its isz reads the shared counter, but does not write it.

But H_2 does not SIM-commute in H, so no scalable implementation is implied—and, in fact, none is possible. The problem is that the caller can reason about order via the dec return values. Only a degenerate implementation, such as one that refused to respond to certain requests, could avoid tracking this order in a nonconflict-free way.

We can make dec commute by eliminating its return value. If we modify the specification so that inc and dec return nothing, then any region consisting exclusively of these operations commutes in any history. A version of H with this modified specification is

$$H' = [\mathsf{reset}(2)_3, \overline{\mathsf{OK}}_3, \underbrace{\mathrlap{\overbrace{\phantom{\mathsf{isz}_1, \overline{\mathsf{NZ}}_1, \mathsf{isz}_2, \overline{\mathsf{NZ}}_2, \mathsf{dec}_3, \overline{\mathsf{OK}}_3}}^{H'_3}}\underbrace{\mathsf{isz}_1, \overline{\mathsf{NZ}}_1, \mathsf{isz}_2, \overline{\mathsf{NZ}}_2}_{H'_1}, \underbrace{\mathsf{dec}_3, \overline{\mathsf{OK}}_3, \mathsf{dec}_4, \overline{\mathsf{OK}}_4}_{H'_2}}_{H'_4}].$$

H'_2, unlike H_2, SIM-commutes, so there must be an implementation that is conflict-free there. Per-thread counters give us such an implementation: each dec can modify its local counter, while isz sums the per-thread values. Per-thread and per-core sharding of data structures like this is a common and long-standing pattern in scalable implementations.

The rule highlights at least one more opportunity in this history. H'_3 also SIM-commutes in H. However, the per-thread counter implementation is not conflict-free for H'_3: dec_3 will write one component of the state that is read and summed by isz_1 and isz_2. But, again, there is a conflict-free implementation based on adding a Boolean "zeroness" snapshot as well as per-thread counters. isz simply returns this snapshot. When dec reduces a per-thread value to zero or below, it reads and sums all per-thread values and updates the snapshot if necessary.

3.5. Discussion

The rule pushes state and history dependence to an extreme: it makes a statement about a *single* history. In broadly commutative interfaces, the arguments and system states for which a set of operations commutes often collapse into fairly well-defined classes (e.g., file creation might commute whenever the containing directories are different). In practice, implementations scale for whole classes of states and arguments, not just for specific histories.

On the other hand, there can be limitations on how broadly an implementation can scale. It is sometimes the case that a set of operations commutes in more than one class of situation, but no single implementation can scale for all classes. For instance, in our modified reference counter, H'_1, H'_2, and H'_3 all SIM-commute in H', and we described a scalable implementation for each situation. However, H'_4 does not SIM-commute, even though it is a union of SIM-commutative pieces: if the two dec operations were reordered to the start of the region, then the isz operations would have to return different values. Any reasonable counter implementation must fail to scale in H'_4, because isz must return different values depending on whether it ran before or after the dec invocations, and this requires communication between the cores that ran dec and isz. This can be proved using a converse of the rule: when a history contains a non-SIM-commutative region, no non-degenerate implementation can be scalable in that region.[6] (The non-degeneracy condition eliminates implementations that, for example, never respond to any invocation, or always respond with an error return value.)

In our experience, real-world interface operations rarely demonstrate such mutually exclusive implementation choices. For example, the POSIX implementation in Section 5 scales quite broadly, with only a handful of cases that would require incompatible implementations.

4. DEFINING COMMUTATIVE INTERFACES

This section demonstrates more situations of interface-level reasoning enabled by the rule, using POSIX, the standard interface for Unix-like operating systems.

The following sections explore four general classes of changes that make POSIX operations commute in more situations, enabling more scalable implementations.

4.1. Decompose compound operations

Many POSIX APIs combine several operations into one, limiting the combined operation's commutativity. For example, fork both creates a new process and snapshots the current process's entire memory state, file descriptor state, signal mask, and several other properties. As a result, fork fails to commute with most other operations in the same process, including memory writes, address space operations, and many file descriptor operations. However, applications often follow fork with exec, which undoes most of fork's suboperations. With only fork and exec, applications are forced to accept these unnecessary suboperations that limit commutativity. POSIX has a posix_spawn call that addresses this problem by creating a process and loading an image directly (CreateProcess in Windows is similar). This is equivalent to fork followed by exec, eliminating the need for many of fork's suboperations. As a result, posix_spawn commutes with most other operations and permits a broadly scalable implementation.

Another example, stat, retrieves and returns many different attributes of a file simultaneously, which makes it noncommutative with operations on the same file that change any attribute returned by stat (such as link, chmod, chown, write, and even read). In practice, applications invoke stat for just one or two of the returned fields. An alternate API that gave applications control of which field or fields were returned would commute with more operations and enable a more scalable implementation of stat.[6]

POSIX has many other examples of compound return values. sigpending returns all pending signals, even if the caller only cares about a subset; and select returns all ready file descriptors, even if the caller needs only one of them.

4.2. Embrace specification nondeterminism

POSIX requires that the open system call returns the lowest-numbered unused file descriptor (FD) for the newly opened file. This rule is a classic example of overly deterministic design that results in poor scalability. Because of this rule, open operations in the same process (and any other FD allocating operations) do not commute, since the order in which they execute determines the returned FDs. This constraint is rarely needed by applications, and an alternate interface that could return any unused FD could use scalable allocation methods, which are well-known. Many other POSIX interfaces get this right: mmap can return any unused virtual address and creat can assign any unused inode number to a new file.

4.3. Permit weak ordering

Another common source of limited commutativity is strict ordering requirements between operations. For many operations, ordering is natural and keeps interfaces simple to use; for example, when one thread writes data to a file, other threads can immediately read that data. Synchronizing operations like this are naturally noncommutative. Communication interfaces, on the other hand, often enforce strict ordering, but may not need to. For instance, most systems order all messages sent via a local Unix domain socket, even when using SOCK_DGRAM, so any send and recv system calls on the same socket do not commute (except in error conditions). This is often unnecessary, especially in multi-reader or multi-writer situations, and an alternate interface that does not enforce ordering would allow send and recv to commute as long as there is both enough free space and enough pending messages on the socket. This in turn would allow an implementation of Unix domain sockets to support scalable communication.

4.4. Release resources asynchronously

A closely related problem is that many POSIX operations have global effects that must be visible before the operation returns. This is generally good design for usable interfaces, but for operations that release resources, this is often stricter than applications need and expensive to ensure. For example, writing to a pipe must deliver SIGPIPE immediately if there are no read FDs for that pipe, so pipe writes do not commute with the last close of a read FD. This requires aggressively tracking the number of read FDs; a relaxed specification that promised to eventually deliver the SIGPIPE would allow implementations to use more scalable read FD tracking. Similarly, munmap does not commute with memory reads or writes of the unmapped region from other threads, because other threads should not be able to write to the unmapped region after munmap returns (even though depending on this behavior usually indicates a bug). Indeed, enforcing this requires remote TLB shootdowns, which do not scale on today's hardware. An munmap (or an madvise) that released virtual memory asynchronously would let the kernel reclaim physical memory lazily and batch or eliminate remote TLB shootdowns.

As another example, to build a scalable reference counter, we start with the interface described in Section 3.4: inc and dec both return nothing and hence always commute. In place of the isz operation, we introduce a new review operation that finds all objects whose reference counts recently reached zero; this frees the developer from having to periodically call isz on their own. review does not commute in any sequence where *any* object's reference count has reached zero and its implementation conflicts on a small number of cache lines even when it does commute. However, unlike dec, the user can choose how often to invoke review. More frequent calls clean up freed memory more quickly, but cause more conflicts. In our implementation of this scheme, called Refcache,[7] review is called at 10 ms intervals, which is several orders of magnitude longer than the time required by even the most expensive conflicts on current multicores.

5. DESIGNING FOR CONFLICT-FREEDOM

To evaluate the implementation difficulty of the previous section's commutative interfaces, we built sv6, a research operating system that aims to provide a POSIX-like interface with as much scalability as is reasonably possible. sv6 includes a ramfs-like in-memory file system called ScaleFS[8] and a virtual memory system called RadixVM.[7] In designing and implementing sv6, the rule told us that conflict-free implementations were possible in many cases, which forced us to come up with designs that achieved conflict-freedom. Without the rule, we would have given up too soon, deciding that some corner cases simply cannot be made to scale.

Problems in achieving conflict-freedom fell into two broad categories. On the one hand, we found situations where a single logical object (such as a reference counter, a pool of memory, or the scheduler queue) was accessed from many cores. Here, we typically used per-core data structures for the commutative parts of the API, and tried to ensure that noncommutative parts of the API (such as reconciling per-core reference counts, or stealing free memory pages or runnable threads from other cores when one core runs out) are invoked rarely and minimize cache-line movement when they are invoked. In some cases this required designing new algorithms, such as Refcache.

On the other hand, we also encountered situations that accessed logically distinct objects (e.g., files in a directory, or pages in a virtual address space), but the data structures typically used to access these objects induced unnecessary conflicts. In particular, we discovered that many sophisticated data structures like red-black trees, splay trees, AVL trees, concurrent lock-free skip lists, etc., are a poor fit for the scalable commutativity rule. For example, balancing operations on binary trees have nonlocal effects: an operation on one branch can cause conflicts over much of the tree. Lock-free skip lists and other lock-free balanced lookup data structures avoid locking, but still induce conflicts on operations that should commute: inserts and removes make nonlocal memory writes to preserve balance (or an equivalent), and those writes conflict with commutative lookups. The effect of these conflicts on performance can be dramatic. A frequent solution involved switching to array-based data structures, which tend to naturally lend themselves to avoiding conflicts for commutative operations. For example, using an array to represent the open file descriptors for a process naturally provides conflict-freedom for operations on distinct file descriptors, because those operations access different addresses in the array.

Naive arrays are not great for situations where the key space is large. One solution for medium-size keys is to use a radix tree. For instance, we use radix trees in the sv6 virtual memory system, RadixVM,[7] to implement the mapping from virtual addresses to the corresponding mapped objects. Since radix trees have no balancing operations, accesses to different addresses tend to not conflict. At the same time, simple compression techniques in the radix tree allow for a compact representation that's much more efficient than a single flat array.

For large or variable-sized keys, hash tables are a natural choice. For example, in the sv6 file system, we use a hash table to represent each directory. This means that concurrent operations on different file names in a single directory are unlikely to conflict (unless they map to the same hash table bucket). This is in contrast to traditional file system designs that take out a single lock to ensure that operations do not modify the same directory entry at the same time.

6. TESTING FOR CONFLICT-FREEDOM

Fully understanding the commutativity of a complex interface is tricky, and checking if an implementation achieves conflict-freedom whenever operations commute adds another dimension to an already difficult task. To help developers apply the rule during testing, we developed a tool called COMMUTER that automates this process.[6] First, COMMUTER takes a symbolic model of an interface and computes precise conditions for when that interface's operations commute. Second, COMMUTER uses these conditions to generate concrete tests of sets of operations that commute according to the interface model, and thus should have a conflict-free implementation according to the commutativity rule. Third, COMMUTER checks whether a particular implementation is conflict-free for each test case. A developer can use these test cases to understand the commutative cases they should consider, to iteratively find and fix scalability bottlenecks in their code, and to perform regression tests to ensure scalability bugs do not creep into the implementation over time.

To illustrate how COMMUTER can help with testing for scalability, we wrote a symbolic model of the POSIX interface covering file system and virtual memory operations, and checked the resulting test cases against Linux and the sv6 operating system. The results are shown in Figures 2 and 3, respectively. Each square represents a pair of system calls. The color of each square represents the fraction of test cases that fail to be conflict-free despite being commutative.

In the case of Linux, we can see that the kernel is already quite scalable: many pairs of system calls are conflict-free for all tests generated by COMMUTER. However, there are also many pairs that commute but are not conflict-free. This indicates that even a mature and reasonably scalable operating system implementation misses many cases that can be made

Figure 2. Conflict-freedom of commutative system call pairs in Linux 3.8, showing the fraction and absolute number of test cases generated by COMMUTER that are *not* conflict-free for each system call pair.

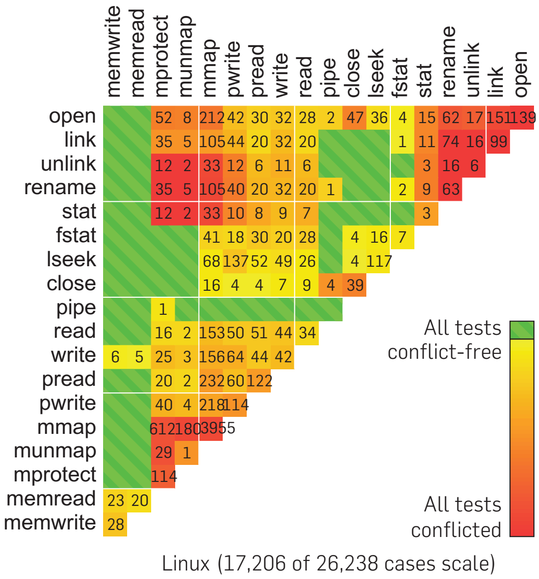
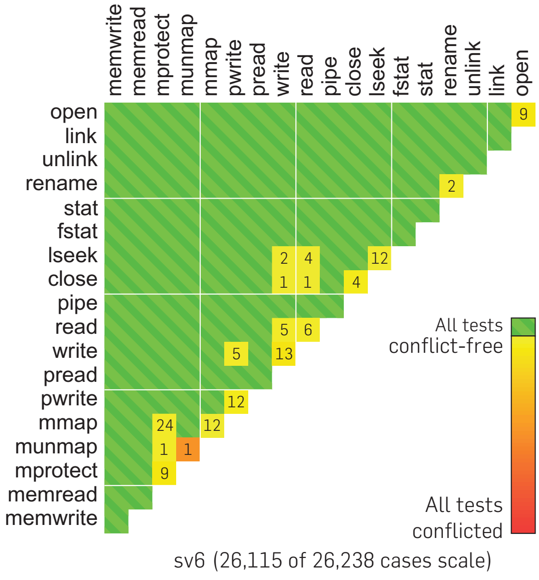


Figure 3. Conflict-freedom of commutative system call pairs in sv6.



to scale according to the commutativity rule. Some of these correspond to well-known scalability problems in Linux, such as concurrent operations on different file names in the same directory (which conflict on a per-directory lock) or concurrent operations on the virtual memory subsystem (which conflict on a per-address-space lock[7]). Others are new bottlenecks that may not have been previously discovered: COMMUTER has systematically discovered latent scalability problems.

In contrast with Linux, sv6 is conflict-free for nearly every commutative test case. In part this is due to our choice of data structures that are naturally conflict-free, as described in the previous section. While testing sv6, COMMUTER also discovered many commutative corner cases that we would not have thought of by ourselves. For example, consider the rename system call and the access system call, which can be used to check if a file exists. Suppose there are two existing files, a and b. COMMUTER discovered that rename(a, b) commutes with access(b), because in either order, rename succeeds and access indicates that b exists. However, our initial implementation was not conflict-free, because access used an internal function that not only checked if the file exists, but also looked up the file's inode. To make this case conflict-free, we introduced a separate function to check whether a file name exists in a directory hash table, without actually reading its corresponding value. During testing, we discovered a number of other common design patterns, such as deferring work whenever possible, preceding pessimism (i.e., writes to memory locations) with optimistic read-only checks, and avoiding reads unless absolutely necessary.

For a small number of commutative operations, sv6 is not conflict-free. The majority of these cases involve idempotent updates to internal state, such as two lseek operations that both seek a file descriptor to the same offset, or two anonymous mmap operations with the same fixed base address and permissions. While it is possible to implement these scalably, every implementation we considered significantly reduced the performance of more common operations, so we explicitly chose to favor common-case performance over total scalability. Other cases represent intentional engineering decisions in the interest of practical constraints on memory consumption and sequential performance. Complex software systems inevitably involve conflicting requirements, and scalability is no exception. However, the presence of the rule forced us to explicitly recognize, evaluate, and justify where we made such trade-offs.

7. DISCUSSION

One surprising aspect of the rule is that it allows us to reason about scalability without having to measure the throughput of a system as a function of the number of cores. Indeed, this paper contains no such graph. To be sure that our rule works in practice, we measured the scalability of a mail server running on sv6, using commutative system calls. The result was perfect scalability. On the one hand, this demonstrates the power of the rule: even for a previously untested hardware system and workload, we are able to confidently predict scalability. On the other hand, scalability is not the same as performance, and a perfectly scalable implementation could have lower total performance than an implementation tuned for efficiency on a small number of cores.

One potential way to expand the reach of the rule and create more opportunities for scalable implementations is to find ways in which *nonconflict-free* operations can scale. For example, while streaming computations are in general not linearly scalable because of interconnect and memory contention, we have had success with scaling interconnect-aware streaming computations. These computations place threads on cores so that the structure of sharing between threads matches the structure of the hardware interconnect and such that no link is oversubscribed. On one 80-core x86 system, repeatedly shifting tokens around a ring mapped to the hardware interconnect achieves the same throughput regardless of the number of cores in the ring, even though every operation causes conflicts and communication. Mapping computations to this model might be difficult, and given the varying structures of multicore interconnects, the model itself may not generalize. However, this problem has close ties to job placement in data centers and may be amenable to similar approaches. Likewise, the evolving structures of data center networks could inform the design of multicore interconnects that support more scalable computations.

8. RELATED WORK

This section briefly explains the relation between the scalable commutativity rule and previous work that explores thinking about scalability and commutativity. For a more in-depth discussion of related work that also covers scalable operating systems and testing approaches we refer the reader to Clements's thesis.[6]

8.1. Scalability

Israeli and Rappoport[12] introduce the notion of disjoint-access-parallel memory systems. Roughly, if a shared memory system is disjoint-access-parallel and a set of processes access disjoint memory locations, then those processes scale linearly. Like the commutativity rule, this is a conditional scalability guarantee: if the application uses shared memory in a particular way, then the shared memory implementation will scale. However, where disjoint-access parallelism is specialized to the memory system interface, our work encompasses any software interface. Attiya et al.[3] extend Israeli and Rappoport's definition to additionally require non-disjoint reads to scale. Our work builds on the assumption that memory systems behave this way and we have confirmed that real hardware closely approximates this behavior.[6]

Both the original disjoint-access parallelism paper and subsequent work[18] explore the scalability of processes that have some amount of non-disjoint sharing, such as compare-and-swap instructions on a shared cache line or a shared lock. Our work takes a black-and-white view because we have found that, on real hardware, a single modified shared cache line can wreck scalability.

The Laws of Order[2] explore the relationship between the "strong noncommutativity" of an interface and whether any implementation of that interface must contain atomic and/or fence instructions for correct concurrent execution. These instructions slow down execution by interfering with out-of-order execution, even if there are no memory access conflicts. The Laws of Order resemble the commutativity rule, but draw conclusions about sequential performance, rather than scalability.

It is well understood that cache-line contention can result in bad scalability. A clear example is the design of the MCS lock,[14] which eliminates scalability collapse by avoiding contention for a particular cache line. Other good examples include scalable reference counters.[1, 5, 9] The commutativity rule builds on this understanding and identifies when arbitrary interfaces can avoid conflicting memory accesses.

8.2. Commutativity

The use of commutativity to increase concurrency has been widely explored. Steele describes a parallel programming discipline in which all operations must be either causally related or commutative.[21] His work approximates commutativity as conflict-freedom. We show that commutative operations always have a conflict-free implementation, implying that Steele's model is broadly applicable. Rinard and Diniz[17] describe how to exploit commutativity to automatically parallelize code. They allow memory conflicts, but generate synchronization code to ensure atomicity of commutative operations. Similarly, Prabhu et al.[16] describe how to automatically parallelize code using manual annotations rather than automatic commutativity analysis. Rinard and Prabhu's work focuses on the *safety* of executing commutative operations concurrently. This gives operations the opportunity to scale, but does not ensure that they will. Our work focuses on scalability directly: we show that any concurrent, commutative operations have a scalable implementation.

The database community has long used logical readsets and writesets, conflicts, and execution histories to reason about how transactions can be interleaved while maintaining serializability.[4] Weihl extends this work to abstract data types by deriving lock conflict relations from operation commutativity.[22] Transactional boosting applies similar techniques in the context of software transactional memory.[10] Shapiro et al.[19, 20] extend this to a distributed setting, leveraging commutative operations in the design of replicated data types that support updates during faults and network partitions. Like Rinard and Prabhu's work, the work in databases and its extensions focuses on the safety of executing commutative operations concurrently, not directly on scalability.

9. CONCLUSION

The scalable commutativity rule helps developers to reason about scalability in all three phases of software design: defining an interface, designing and implementing the software, and testing its scalability properties. The rule does not require the developer to have a target workload or a physical machine to reason about scalability. We hope that programmers will find the commutativity rule helpful in producing software that is scalable by design.

Acknowledgments

This research was supported by NSF awards SHF-964106 and CNS-1301934, by Quanta, and by Google. Eddie Kohler was partially supported by a Microsoft Research New Faculty Fellowship and a Sloan Research Fellowship.

References

1. Appavoo, J., da Silva, D., Krieger, O., Auslander, M., Ostrowski, M., Rosenburg, B., Waterland, A., Wisniewski, R.W., Xenidis, J., Stumm, M., Soares, L. Experience distributing objects in an SMMP OS. *ACM Trans. Comput. Syst. 25*, 3 (August 2007).
2. Attiya, H., Guerraoui, R., Hendler, D., Kuznetsov, P., Michael, M.M., Vechev, M. Laws of order: Expensive synchronization in concurrent algorithms cannot be eliminated. In *Proceedings of the 38th ACM Symposium on Principles of Programming Languages* (Austin, TX, January 2011), 487–498.
3. Attiya, H., Hillel, E., Milani, A. Inherent limitations on disjoint-access parallel implementations of transactional memory. In *Proceedings of the 21st Annual ACM Symposium on Parallelism in Algorithms and Architectures* (Calgary, Canada, August 2009), 69–78.
4. Bernstein, P.A., Goodman, N. Concurrency control in distributed database systems. *ACM Comput. Surv. 13*, 2 (June 1981), 185–221.
5. Boyd-Wickizer, S., Clements, A., Mao, Y., Pesterev, A., Kaashoek, M.F., Morris, R., Zeldovich, N. An analysis of Linux scalability to many cores. In *Proceedings of the 9th Symposium on Operating Systems Design and Implementation (OSDI)* (Vancouver, Canada, October 2010).
6. Clements, A.T. The scalable commutativity rule: Designing scalable software for multicore processors. PhD thesis, Massachusetts Institute of Technology (June 2014).
7. Clements, A.T., Kaashoek, M.F., Zeldovich, N. RadixVM: Scalable address spaces for multithreaded applications (revised 2014-08-05). In *Proceedings of the ACM EuroSys Conference* (Prague, Czech Republic, April 2013), 211–224.
8. Clements, A.T., Kaashoek, M.F., Zeldovich, N., Morris, R.T., Kohler, E. The scalable commutativity rule: Designing scalable software for multicore processors. *ACM Trans. Comput. Syst. 32*, 4 (January 2015), 10:1–10:47.
9. Ellen, F., Lev, Y., Luchango, V., Moir, M. SNZI: Scalable nonzero indicators. In *Proceedings of the 26th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (Portland, OR, August 2007), 13–22.
10. Herlihy, M., Koskinen, E. Transactional boosting: A methodology for highly-concurrent transactional objects. In *Proceedings of the 13th ACM Symposium on Principles and Practice of Parallel Programming* (Salt Lake City, UT, February 2008), 207–216.
11. Herlihy, M.P., Wing, J.M. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Programm. Lang. Syst. 12*, 3 (1990), 463–492.
12. Israeli, A., Rappoport, L. Disjoint-access-parallel implementations of strong shared memory primitives. In *Proceedings of the 13th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (Los Angeles, CA, August 1994), 151–160.
13. McKenney, P.E. Differential profiling. *Softw. Pract. Exp. 29*, 3 (1999), 219–234.
14. Mellor-Crummey, J.M., Scott, M.L. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Trans. Comput. Syst. 9*, 1 (1991), 21–65.
15. Papamarcos, M.S., Patel, J.H. A low-overhead coherence solution for multiprocessors with private cache memories. In *Proceedings of the 11th Annual International Symposium on Computer Architecture* (Ann Arbor, MI, June 1984), 348–354.
16. Prabhu, P., Ghosh, S., Zhang, Y., Johnson, N.P., August, D.I. Commutative set: A language extension for implicit parallel programming. In *Proceedings of the 2011 ACM SIGPLAN Conference on Programming Language Design and Implementation* (San Jose, CA, June 2011), 1–11.
17. Rinard, M.C., Diniz, P.C. Commutativity analysis: A new analysis technique for parallelizing compilers. *ACM Trans. Programm. Lang. Syst. 19*, 6 (November 1997), 942–991.
18. Roy, A., Hand, S., Harris, T. Exploring the limits of disjoint access parallelism. In *Proceedings of the 1st USENIX Workshop on Hot Topics in Parallelism* (Berkeley, CA, March 2009).
19. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M. Conflict-free replicated data types. In *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems* (Grenoble, France, October 2011), 386–400.
20. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M. Convergent and commutative replicated data types. *Bull. EATCS 104* (June 2011), 67–88.
21. Steele, G.L., Jr. Making asynchronous parallelism safe for the world. In *Proceedings of the 17th ACM Symposium on Principles of Programming Languages* (San Francisco, CA, January 1990), 218–231.
22. Weihl, W.E. Commutativity-based concurrency control for abstract data types. *IEEE Trans. Comput. 37*, 12 (December 1988), 1488–1505.

Austin T. Clements, M. Frans Kaashoek, Robert T. Morris, and Nickolai Zeldovich ({aclements, kaashoek, rtm, zeldovich}@csail.mit.edu), MIT CSAIL, Cambridge, MA.

Eddie Kohler (kohler@seas.harvard.edu), Harvard University School of Engineering and Applied Sciences, Computer Science Area, Cambridge, MA.

DOI:10/1145/3068903

To view the accompanying paper, visit doi.acm.org/10.1145/3068766 rh

Technical Perspective
Linking Form, Function, and Fabrication

By Helmut Pottmann

"COMPUTER GRAPHICS ARE pictures and movies created using computers." This opening sentence from Wikipedia's entry on computer graphics becomes increasingly outdated as graphics is about to close the loop between virtual and physical reality and from digital design to fabrication. Is this a new trend? Not quite, but the magnitude of the current development and the potential impact may be bigger than ever before.

Geometric modeling, a subfield of computer graphics, has been motivated by industrial needs at the advent of computer-aided manufacturing, aiming at increased productivity via a completely digital workflow from design to production. Research in geometric modeling has been highly successful in creating a huge variety of shape modeling functionalities for 3D-design systems. The possibilities for digital shape design are almost unlimited and highly effective for the creation of "pictures and movies." But did the field achieve the goals toward manufacturing? I am not convinced here, since purely geometry-driven shape modeling creates bottlenecks when moving toward engineering and fabrication.

Some of the problems are related to *geometry representations*. B-splines and subdivision schemes proved to be highly effective for freeform shape modeling, but the resulting digital models are not yet suitable for simulation. There are various reasons for this: The models are usually not watertight and need to be "repaired" in a time-consuming laborious process; for example, at surface/surface intersections. Moreover, the models are surface based. For simulation, one requires meshes, not just for bounding surfaces, but also for the interior. For the actual production, which often employs CNC machining of molds, there are further conversion issues since the machines are not capable of precisely following splines.

There are various efforts to change this picture: a prominent reaction from mathematics is "*isogeometric analysis*," which eliminates re-meshing by using the same spline-based representation for both modeling and simulation. However, the result of a simulation may reveal a design problem; for example, a mismatch of form and function, or the presence of geometry, which contradicts a certain manufacturing process or material behavior, requiring changes to the design. To avoid such costly feedback loops between design, engineering, and fabrication, *research in computer graphics has recently tried to incorporate key aspects of function and fabrication into an "intelligent" shape modeling process.* This is not easy at all, since one wants to achieve interactivity of the modeling tools, but at the same time satisfy numerous constraints.

A great example for this trend is depicted in the following paper—it couples shape and function and results in unexpected, almost miraculous behavior of objects. In the present case, these objects are toys, namely spinning tops or yo-yos of nearly arbitrary shape. However, the research is far deeper. It merges the previously mentioned research focus with another hot topic of research: *Additive manufacturing (AM)*. With AM, geometric complexity comes almost for free. Here, complexity is not restricted to the outer surface, but also applies to the interior of an object. This is in perfect harmony with the goals of the highlighted article: Compute and fabricate the interior of an object so that it possesses perfect spinning properties. Mathematically, one "just" has to get right a few integrals over the entire body, namely the ones that determine barycenter and moments of inertia. While this can be nicely cast into an algorithm, one cannot ask a designer to care about such properties. It is basically impossible to model a nonrotationally symmetric object with great spinning behavior with any of the available modeling systems. We have here an excellent example for "*computational fabrication*." Computation not only accelerates the design and fabrication process, but in fact is the only way of creating such objects.

This is a step in a new direction, namely the exploitation of the nearly unlimited design space for objects fabricated with AM. Computer graphics is about to take a leading role in AM research, including the authors of the following paper. I just mention here work on "geometric materials," which act in a surprising, predefined way, or on the design of musical instruments of unconventional shape. A related classical topic is "topology optimization," where one optimizes the interior of objects so that minimal material usage results in sufficient strength to resist predefined loads.

Finally, let me point to work on the much larger architectural scale. A recent direction of research, called "*architectural geometry*," aims at making geometrically complex architectural structures affordable through novel computational tools by linking design, function, and fabrication. Substantial contributions to this field, for example, on self-supporting freeform structures, have also been made by some of the authors of the following paper (in cooperation with Philippe Block; http://www.block.arch.ethz.ch/).

Computer graphics is a field where technology meets design. We are in strong need for this type of research, on all scales, from the micro-level of material behavior to the macro-level of buildings or even entire cities. C

Helmut Pottmann is a professor of applied geometry at Technische Universität Wien, Vienna, Austria.

DOI:10.1145/3068766

Spin-It: Optimizing Moment of Inertia for Spinnable Objects

By Moritz Bächer, Bernd Bickel, Emily Whiting, and Olga Sorkine-Hornung

Abstract

Spinning tops and yo-yos have long fascinated cultures around the world with their unexpected, graceful motions that seemingly elude gravity. Yet, due to the exceeding difficulty of creating stably spinning objects of asymmetric shape in a manual trial-and-error process, there has been little departure from rotationally symmetric designs. With modern 3D printing technologies, however, we can manufacture shapes of almost unbounded complexity at the press of a button, shifting this design complexity toward computation.

In this article, we describe an algorithm to generate designs for spinning objects by optimizing their mass distribution: as input, the user provides a solid 3D model and a desired axis of rotation. Our approach then modifies the interior mass distribution such that the principal directions of the moment of inertia align with the target rotation frame. To create voids inside the model, we represent its volume with an adaptive multiresolution voxelization and optimize the discrete voxel fill values using a continuous, nonlinear formulation. We further optimize for rotational stability by maximizing the dominant principal moment. Our method is well-suited for a variety of 3D printed models, ranging from characters to abstract shapes. We demonstrate tops and yo-yos that spin surprisingly stably despite their asymmetric appearance.

1. INTRODUCTION

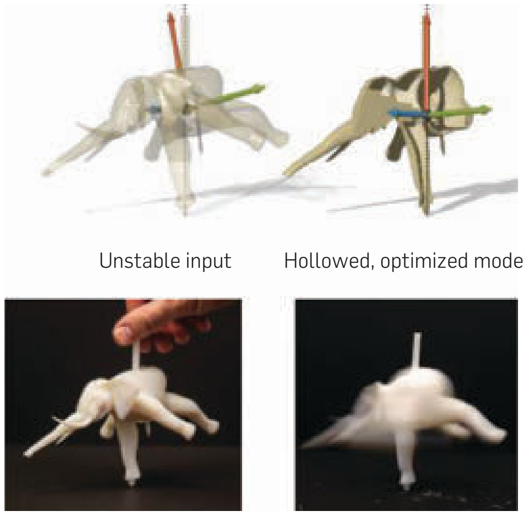
Spinning toys have existed since antiquity as playful objects that capture the imagination. Invented independently all over the world, spinning tops are referenced in ancient Greek literature,[12] and evidence of clay tops has been found in ancient cities dating as early as 3500 B.C. Similarly, while yo-yos are believed to have been invented in China, there are many historical references, including in Mozart's *The Marriage of Figaro* where a yo-yo is spun to relieve stress.[17] Despite the long tradition of these toys, until today creating new designs has been a trial-and-error process, calling on the intuition and meticulous patience of artists and hobbyists. Moreover, there has been little departure from rotationally symmetric designs.

Much attention has been devoted in the field of classical mechanics to *explaining* the motion of spinning objects; however, the focus has been primarily on analysis[8, 9, 19, 21] rather than design. In this article, we investigate the unique geometric properties of shapes that spin, with an eye on digital modeling and free-form design. A stable spin has requirements on rotational inertia, including precise positioning of the center of mass and correct alignment of the primary axes of the body. We propose an algorithm to optimize for these inertial properties, for example, to design a spinning top that rotates smoothly and stably and can be fabricated using 3D printing.

In our approach, users provide an initial design for a spinning model, specified as a 3D surface mesh. Along with the input geometry, the user may specify the desired axis of spinning and the contact point with the support. The mass distribution is then optimized to ensure that the primary axis for the moment of inertia aligns with the desired axis of rotation. Since the moment of inertia depends on the entire volume, rather than just on the surface geometry, we preserve the appearance of the input design by changing the internal mass distribution as we illustrated in Figure 1 on an elephant top.

We first formulate a nonlinear functional that measures the *spinnability* of a solid shape about a user-defined axis. Using this measure, we then devise constrained optimization problems that align the principal axes for moment of inertia with user-specified rotation axes. To this end, we maximize the ratio of principal moments in the dominant and lateral directions and place the center of mass on the rotation axis. For our tops, we further improve stability by lowering the center of mass, simultaneously reducing the mass.

Figure 1. We describe an algorithm for the design of spinning tops and yo-yos. Our method optimizes the inertia tensor of an input model by changing its mass distribution, allowing long and stable spins even for complex, asymmetric shapes.



Unstable input

Hollowed, optimized model

Our spinning top design

Elephant in motion

The original version of this paper was published in *Proceedings of SIGGRAPH'14*, August 2014, ACM.

Our approach is effective on a wide range of models, from characters to abstract geometric forms. We employ an adaptive octree for discretizing the fill volume of our input shapes and validate our results by fabricating the optimized shapes; the objects can be stably spun despite their complex, asymmetric exterior appearance.

2. RELATED WORK

Fabrication-oriented design. Fabrication-oriented design has gained increasing interest from the computer graphics community, triggered by advances in 3D manufacturing technology. Various physical properties have been explored in this shape modeling context including deformation properties,[4, 22] articulation behavior,[1, 5] structural strength,[23, 24] and kinematic structures.[6, 7, 25]

Most related to our effort is the work by Prévost et al.[20]: they proposed an approach for balancing static models at rest that applies a combination of voxel carving and deformation to control the center of mass. Our work addresses a more general problem of stability under rotational motion, involving both center of mass and moment of inertia. While Prévost et al.[20] use a plane sweeping heuristic for carving, we solve our constrained combinatorial problems by recasting them as sequential linear-quadratic programs using relaxation on the fill variables. As discussed later, our optimization can be used for static balancing, tending to find more stable solutions.

Rotational dynamics design. Furuta et al.[10] combine a geometric modeling interface and a rigid body simulator for the design of kinetic art, providing real-time previews of the resulting motion during the design process. While restricted to forward simulations, this approach allows the user to quickly explore many trial-and-error experiments. We avoid trial-and-error and simulation, directly estimating models from user-specified geometries.

Hirose et al.[13] enforce symmetries along with additional geometric constraints to create sphericons. In contrast, we do not require a feasible starting solution and do not incorporate constraints other than the ones prescribed by the desired physical properties, enabling free-form design. To the best of our knowledge, we are the first to study the computational design of spinning toys with asymmetric appearance.

Moment of inertia. Moment of inertia is a fundamental property of rigid bodies. It specifies the required torque needed for a change in angular velocity and is, for example, an essential component in physics-based animation for rigid body simulations or dynamics and control of characters.[16] Design for moment of inertia has been investigated in mechanical engineering. However, the methods and objectives used differ significantly. Our approach further generalizes to free-form shapes and we formulate an exact energy and derivatives.

Topology optimization. Topology optimization methods solve engineering problems of distributing a limited amount of material in a design space.[3] While our adaptive voxel discretization shares similarities with solution techniques common in this field, spinnability properties have not been considered by prior work.

3. FUNDAMENTALS AND OVERVIEW

Given a 3D shape, we aim to generate spinnable models by altering their mass distribution, while keeping the appearance as close to the original as possible. In the following sections, we describe the user input, fundamental mass properties, and spinnability metrics needed to optimize the input toward a stably rotating object.

3.1. User input

The user provides the surface of a solid 3D shape, along with the desired spinning axis **a**. The axis origin is set to the contact point **p** as shown in Figure 2a, which can be user-defined or chosen as the lowest point on the model w.r.t. the up-direction **a**. For yo-yo designs, the shape is partitioned into two parts and connected with an axle that aligns with **a**, to allow string coiling. The user selects a point **q** on the axle to define the coiling location (Figure 2b).

3.2. Mass properties and constraints

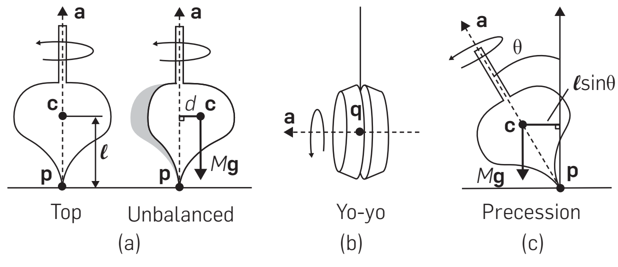
Center of mass. We denote by M the mass of our object and by c the center of mass. If we assume a frictionless spin, the only external torque acting on a spinning top relative to p, is the gravitational torque with magnitude $|\tau| = Mgd$, where g is Earth's gravity and d is the distance from c to the spinning axis (Figure 2a). We constrain the center of mass to lie on the spinning axis so that the net torque on the model around the ground contact point is zero.

Refer to Figure 2c: during the spinning motion, the precession angle θ between the rotational (vertical) and spinning axes increases if the angular velocity ω becomes smaller. We can express the gravitational torque as $|\tau| = Mg\ell\sin\theta$, where ℓ is the height of the center of mass. Hence, we expect a longer, more stable spin for smaller values of ℓ and M.

For yo-yos, the gravitational torque remains zero throughout the spin if we neglect the effect of an uneven coiling of the string.

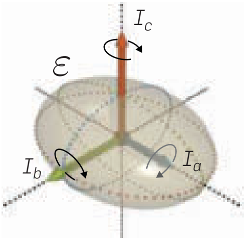
Moment of inertia. Moment of inertia is the analog of mass for rotational motion and measures the resistance to rotations about a given axis. Euler's equations from classical

Figure 2. Spinning yo-yos and tops stably: for spinning tops, the center of mass must lie on the user-specified spinning axis a, otherwise it will cause an unbalanced external torque $|\tau| = Mgd$ relative to p (a). For slower angular velocities, the precession angle θ between rotational (vertical axis) and spinning axes becomes larger (c). For smaller ℓ, the gravitational torque $|\tau| = Mg\ell \sin\theta$ is smaller for equal precession, resulting in a longer spin. For yo-yos, we require the center of mass to coincide with q (b).



mechanics (see, e.g., Ref.[11]) conveniently describe the rotating motion of a rigid body in its body frame, whose axes are the three principal axes of inertia and the origin is **c**. Since there is no external torque acting on the body (for **c** on the spinning axis), we can only spin about an axis with constant angular velocity if it is a principal axis of inertia.

For an arbitrary rigid body, there exists an equivalent ellipsoid with the same inertial properties. We can discuss the preferable axis using an ellipsoid $\mathcal{E}$ with half-axes $\mathbf{h}_a$, $\mathbf{h}_b$, $\mathbf{h}_c$ ($\|\mathbf{h}_c\| \leq \|\mathbf{h}_b\| \leq \|\mathbf{h}_a\|$). Due to symmetry, $\mathcal{E}$'s principal axes of inertia are parallel to its half-axes, and the corresponding moments I_a, I_b, and I_c each equal the sum of squares of the two other half-axes' lengths (omitting a common scale factor), as illustrated in the inset. Hence, the maximal principal axis of inertia corresponds to the shortest axis $\mathbf{h}_c$, and we have $I_c \geq I_b \geq I_a$. If we spin the ellipsoid $\mathcal{E}$ with a constant angular velocity ω about a principal axis of inertia, the kinetic energy K in our system is $\frac{1}{2}I\omega^2$, $I \in \{I_a, I_b, I_c\}$. Since K is proportional to I, we can expect a longer spin for $I = I_c$.



Rotational stability. Rotational stability refers to a body's behavior under small disturbances to its angular velocity ω due to, for example, frictional forces. Given three distinct values for the principal moments of inertia, $I_c > I_b > I_a$, rotation is stable under small perturbations only about the largest and the smallest axis.[11] In the case of two axes having identical principal moments, the rotation is stable only around the distinct axis. For $I_c = I_b = I_a$, no axis is stable, neglecting contact friction. We can observe this effect when trying to spin a marble in place: the orientation of the body changes over time. As long as the condition $I_c > I_b \geq I_a$ holds, we call I_c the dominant and I_b and I_a the lateral principal moments of inertia.

For an asymmetric shape whose maximal principal axis of inertia aligns with the spin (and gravitational) axis and whose moments are distinct $I_c > I_b > I_a$, the top spins stably under the condition[14]:

$$\omega^2 > \frac{Mg\ell}{I_c - I_b}. \tag{1}$$

From this relation we can see that the stability limit depends on the height of the center of mass ℓ and the mass M itself: the lower the centroid and the smaller the mass, the less angular velocity ω is required for a stable spin, confirming our conclusion from the above discussion on precession. Similarly, we need a smaller ω the higher the absolute difference between the largest moment I_c and the mid-moment I_b.

In summary, in order to spin stably, four basic requirements on the mass distribution of the model must be met:

1. The center of mass **c** must lie on axis **a** for spinning tops, or coincide with the axle center **q** for yo-yos.
2. The center of mass **c** should be closer to contact point **p** and the mass M minimal for our tops.
3. The axis **a** should be parallel to the maximal principal axis of inertia.
4. The magnitude of the largest principal moment of inertia should dominate over lateral moments to ensure the stability of the spin.

3.3. Measuring spin quality

To distill the above analysis of spinning properties into a spin quality measure, we formulate energy functionals for our yo-yos and tops. Provided that the basic constraints from Section 3.2 are fulfilled, our functionals assign a spin quality score to a given model $\mathcal{M}$ based on the stability criterion (1). Note that while Equation (1) suggests that a comparison of the mid- and largest moments is sufficient for tops, we consider all moments in our quality measures because the ordering of mid- and smallest axes might flip during our dynamic balancing optimization (see Section 4).

Yo-yos. We measure the spin quality of a yo-yo by summing the squared ratios of the dominant to lateral principal moments of inertia:

$$f_{\text{yo-yo}} = \gamma_{\mathrm{I}} \left[\left(\frac{I_a}{I_c} \right)^2 + \left(\frac{I_b}{I_c} \right)^2 \right], \tag{2}$$

assuming that I_c corresponds to the given spin axis and the center of mass **c** equals the axle center **q**. The function $f_{\text{yo-yo}}$ is our *yo-yo energy functional*; small values correspond to longer, more stable spins.

Tops. To measure the quality of a spinning top, we add a term that penalizes the distance ℓ between the center of mass **c** (which is constrained to lie on the axis **a**) and the contact point **p** and minimizes the mass M, yielding the *top energy functional*:

$$f_{\text{top}} = \gamma_{\mathbf{c}} \left(\ell M \right)^2 + f_{\text{yo-yo}}. \tag{3}$$

The two weights $\gamma_{\mathbf{c}}$ and γ_{I} allow calibrating the relative contributions of the center of mass, inertia and the regularization term of the parameterization that follows (see Section 5).

3.4. Optimizing tops and yo-yos

We turn models into spinnable objects by altering their mass distribution while keeping their appearance unchanged. To this end, we redistribute mass by hollowing the interior with precisely shaped voids. We adopt a multiresolution octree to discretize the interior volume of the object. To generate the voids, we optimize for voxel fill values using a continuous, nonlinear formulation as we discuss in more detail in Section 5. We maximize stability through the energy functionals $f_{\text{yo-yo}}$ (2) and f_{top} (3), respectively.

While hollowing is effective for many models, some special cases over-extend our stability requirements and voids alone cannot accomplish a stable spin. This is due to the nonnegligible material on the object's shell. In the original version of this article,[2] we introduce extensions to our approach, further manipulating mass by either deforming the surface and interior voids or compensating for highly nonoptimal mass distributions with a heavier material in the interior.

4. OPTIMIZING DYNAMIC BALANCE

Before we explain our multiresolution discretization of the interior mass distribution, we formalize our quality measures and requirements from the previous section, casting them as optimization problems in a discretization-independent manner. To evaluate our two quality measures $f_{\text{yo-yo}}$ and f_{top} on a model $\mathcal{M}$ made of a homogeneous material, we need to express its mass properties M, $\mathbf{c}$, and the 3×3 symmetric inertia tensor $\mathbf{I}$.

Assume that the surface $\mathcal{M}$ encloses a region $\Omega \in \mathbb{R}^3$ that corresponds to a solid object with constant density ρ. We express the above quantities using the ten integrals of the monomials of degree ≤ 2 over Ω, collected in a 10-vector:

$$\mathbf{s}_\Omega(\rho) = \left[s_1, s_x, s_y, s_z, s_{xy}, s_{yz}, s_{xz}, s_{x^2}, s_{y^2}, s_{z^2}\right]^T, \qquad (4)$$

$$\text{where} \quad s_t = \rho \int_\Omega t\, dV, \qquad \text{e.g.,} \qquad s_{xy} = \rho \int_\Omega xy\, dV.$$

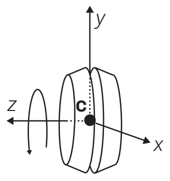
We obtain the following expressions for the mass and center of mass:

$$M = s_1 \quad \text{and} \quad \mathbf{c} = \frac{1}{M}\left[s_x, s_y, s_z\right]^T$$

and $\mathcal{M}$'s inertia tensor:

$$\mathbf{I} = \begin{bmatrix} s_{y^2} + s_{z^2} & -s_{xy} & -s_{xz} \\ -s_{xy} & s_{x^2} + s_{z^2} & -s_{yz} \\ -s_{xz} & -s_{yz} & s_{x^2} + s_{y^2} \end{bmatrix}.$$

Note that we can reduce the volume integrals in $\mathbf{s}_\Omega$ to surface integrals $\mathbf{s}_{\partial\Omega}$ using the Divergence theorem, resulting in analytical expressions for a triangulated surface $\partial\Omega$; see Supplemental Material accompanying the original version of this article Ref.[2]



Coordinate frame for yo-yos. As evident from the formulas above, $\mathbf{c}$ and $\mathbf{I}$ are expressed w.r.t. a coordinate frame. For our yo-yos, the most convenient frame has its origin at the user-provided spin point $\mathbf{q}$ and one of the three axes, say z, points in the direction of the desired spin axis $\mathbf{a}$, as illustrated in the inset. For this choice of frame, the model can only be spun about $\mathbf{a}$ if the center of mass components s_x, s_y, and s_z, and also the off-diagonal elements $-s_{xz}$, $-s_{yz}$ of $\mathbf{I}$ equal zero. Otherwise, $\mathbf{c}$ does not equal $\mathbf{q}$ or the z-axis is not a principal axis of inertia of $\mathcal{M}$. Provided $\mathcal{M}$ fulfills these constraints, $I_z = s_{x^2} + s_{y^2}$ takes on the role of I_c in our functional $f_{\text{yo-yo}}$. If I_x and I_y denote the eigenvalues of the 2×2 upper block

$$\bar{\mathbf{I}} = \begin{bmatrix} s_{y^2} + s_{z^2} & -s_{xy} \\ -s_{xy} & s_{x^2} + s_{z^2} \end{bmatrix},$$

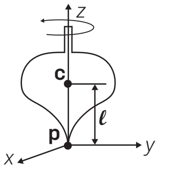
we recall that the trace of $\bar{\mathbf{I}}^2$ is the sum of the squared eigenvalues $I_x^2 + I_y^2$, leading to an elegant reformulation of our yo-yo functional $f_{\text{yo-yo}}$

$$\gamma_{\mathbf{I}} \frac{\operatorname{tr}\left(\bar{\mathbf{I}}^2\right)}{I_z^2} = \gamma_{\mathbf{I}} \frac{\left(s_{y^2} + s_{z^2}\right)^2 + 2s_{xy}^2 + \left(s_{x^2} + s_{z^2}\right)^2}{\left(s_{x^2} + s_{y^2}\right)^2}, \qquad (5)$$

where tr is the trace operator and I_x and I_y take on the roles of I_a and I_b, respectively.

Optimizing yo-yos. To turn an arbitrary model $\mathcal{M}$ into a yo-yo, we therefore need to minimize $f_{\text{yo-yo}}$ with $I_a := I_x$, $I_b := I_y$, and $I_c := I_z$, with the constraints

$$s_t = 0, \qquad \forall t \in \{x, y, z, xz, yz\}. \qquad (6)$$



Parallel Axis Theorem. The body frame centered at $\mathbf{c}$ is not an ideal coordinate system for our tops because the center of mass can move freely along the axis $\mathbf{a}$. A better-suited frame is centered at the contact point $\mathbf{p}$, with the z-axis aligned with $\mathbf{a}$ (see inset). Within this frame, the center of mass $\mathbf{c}$ lies at height ℓ on the z-axis, so that the inertia tensor $\mathbf{I}$ is computed w.r.t. a frame shifted by ℓ w.r.t. our body frame. To evaluate f_{top}, we use the Parallel Axis Theorem, which states that if the axes of two frames are parallel, we can determine the new inertia tensor using the translation vector between the two origins and the body's mass:

$$\mathbf{I}_{CoM} = \mathbf{I} + M\left(\mathbf{c}\mathbf{c}^T - \mathbf{c}^T\mathbf{c}\mathbf{E}\right),$$

where $\mathbf{E}$ is the identity matrix. For our choice of frame, where the center of mass is at $[0, 0, \ell]^T$, the theorem simplifies to

$$\mathbf{I}_{CoM} = \mathbf{I} - \frac{s_z^2}{s_1}\operatorname{diag}(1, 1, 0).$$

Optimizing tops. For our tops, we minimize f_{top}, where I_a and I_b are now the eigenvalues of the upper 2×2 block of the inertia tensor $\mathbf{I}_{CoM}$, and $I_c = s_{x^2} + s_{y^2}$ as before.

Unlike the yo-yo case, $\mathbf{c}$ can move freely on the z-axis. Hence, we relax the equality constraint $s_z = 0$, instead substituting $\ell M = s_z$ in the objective f_{top} (refer to Equation 3).

The constraints to the optimization are then

$$s_t = 0, \qquad \forall t \in \{x, y, xz, yz\}. \qquad (7)$$

Optimizing static balance. Interestingly, the problem of balancing a model at rest is a relaxed version of the top optimization:

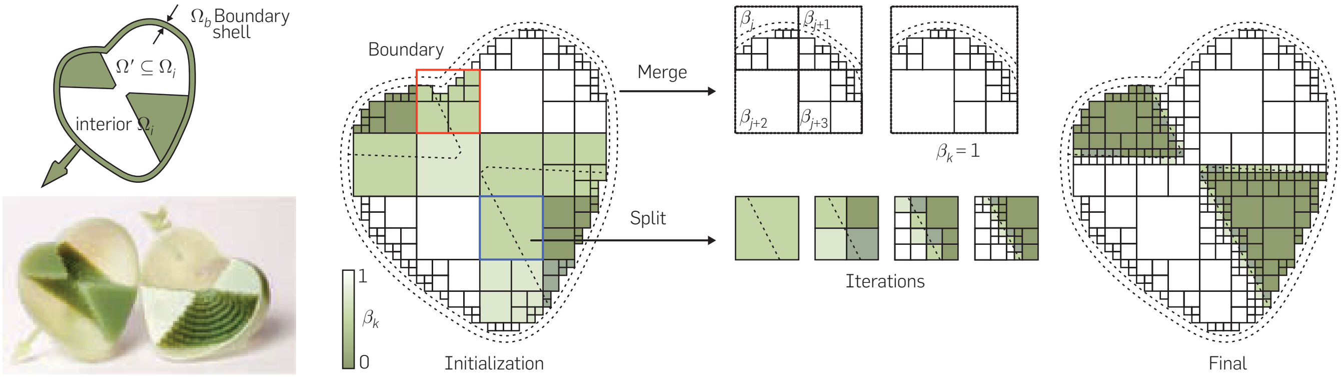
$$\text{minimize } f_{\text{static}} = \gamma_{\mathbf{c}}\ell^2 \quad \text{subject to } s_x = s_y = 0,$$

where we remove the mass term M in f_{static} because only the lowering of $\mathbf{c}$ improves the balance at rest.

5. HOLLOWING

The most nonintrusive way to compensate for unfavorable mass distributions in a model is to introduce voids in the interior, as illustrated in Figure 3. The idea of carving the shape's interior by sweeping a plane through a uniform voxel grid was explored in Prévost et al.[20] for static balancing. We propose a different optimization approach that addresses the inertia tensor in addition to the center of mass, uses a spatially adaptive discretization, and avoids heuristics.

Figure 3. Hollowing: (Left) Our input encloses a volume Ω. By introducing voids Ω′, we can compensate for an unfavorable mass distribution. (Right) To reduce the number of variables and overall time complexity for our voids optimization, we summarize contributions of octree leaf cells in a partition of larger cells shown here for a boundary and an interior cell.



As explained in the previous sections, we aim to minimize $f_{\text{top}}(\Omega)$ subject to the constraints (7), or $f_{\text{yo-yo}}(\Omega)$ subject to the constraints (6). The variable in the optimization is the spatial mass distribution inside the shape, as detailed below. Recall that the functionals and the constraints are expressed in terms of the integrals $\mathbf{s}_{\Omega}$; we explain how these integrals depend on our unknowns.

5.1. Fabrication considerations

We enforce a minimal wall thickness to ensure that the resulting models can be fabricated. As shown in Figure 3 (left), we partition the region Ω into a boundary shell Ω_b and the interior Ω_i, restricting the hollowing to Ω_i. To account for a hollowed region $\Omega' \subseteq \Omega_i$ in our cost functionals, we adjust the volume integrals in Equation (4):

$$\mathbf{s}_{\Omega-\Omega'} = \mathbf{s}_{\Omega} - \mathbf{s}_{\Omega'}.$$

Recall, given an axis, the contribution of a mass element to the moment of inertia is proportional to its squared distance from this axis. Mass on the boundary Ω_b has a high influence on the moment of inertia since it is far from the axis. Therefore, it is desirable for the wall to be as thin as possible within fabrication limits.

5.2. Voxelization

We discretize the interior Ω_i into mass elements Ω_k and optimize a binary fill variable $\beta_k \in \{0, 1\}$ for each, where a value of 1 means that we hollow that element and 0 means we keep it filled. To handle free-form surfaces in our input and provide sufficient degrees of freedom for interior voids, we require our discretization to support fine enough mass elements. One possibility would be to use a high-resolution uniform voxel grid. However, we observe that finest-resolution voxels are only required at the surface separating the void space from the fill and external surface (see, e.g., the interior mass distribution of the Heart in Figure 3, left, bottom). We therefore employ a multiresolution voxelization based on an adaptive octree, thereby significantly reducing the number of fill variables. Our discretized volume integrals then become

$$\mathbf{s}_{\Omega-\Omega'} = \mathbf{s}_{\Omega} - \sum_k \beta_k \mathbf{s}_{\Omega_k},$$

where $\Omega_i = \bigcup_k \Omega_k$ is a partitioning of the interior into octree cells Ω_k. The void space Ω′ consists of all cells Ω_k for which $\beta_k = 1$.

5.3. Optimization approach

Given our adaptive voxel discretization, since the fill values are binary, the resulting minimization problem would be combinatorial. In order to take advantage of continuous optimization techniques, we propose a relaxation approach that allows β_k to take on a continuous value in the interval [0, 1].

The goal of the optimization eventually is to assign binary fill values to each voxel. In practice, we observed that fill variables β_k with a fractional value only occur on the boundary between voids and solid regions. Hence, we sample these regions at a high resolution, ensuring final fractional values correspond to finest resolution cells only (compare with Figure 3, final). Values are then rounded to binary numbers after convergence of the optimization.

This motivates the following optimization algorithm using adaptive refinement (refer to Figure 3, right):

Initialization. We initialize the octree to a mid-level refinement (blue in Figure 3) as a compromise between number of variables and resolution of the initial partitioning. For each cell, we compute $\mathbf{s}_{\Omega_k}$. For cells which overlap the boundary Ω_b (red), we only take the contribution from the volume in Ω_i into account.

Optimization step. We then optimize the fill variables β_k for all cells *k* as explained in detail below.

Split-and-merge. All cells *k* whose fill values are not binary ($\beta_k \in [\varepsilon, 1 - \varepsilon]$) after minimization, are split one level lower if they are not yet at the maximum resolution (see split branch). Conversely, cells with fill values within ε of 0 or 1 are candidates for merging. We merge neighboring cells with the same values into as coarse cells as possible (see merge branch). This gives us a new set of cells Ω_k for which we update $\mathbf{s}_{\Omega_k}$.

Convergence. After each optimization step, and split-and-merge, we check whether all fill values $\beta_k \in [\varepsilon, 1 - \varepsilon]$ correspond to cells Ω_k at the maximum resolution. If so, we terminate the optimization.

Our functionals f_{top} and $f_{\text{yo-yo}}$ are nonlinear in the fill variables β_k. To prevent an underdetermined minimization problem, we penalize differences between fill variables using a uniform symmetric Laplacian **L**, constructed over neighboring cells. This results in the following regularized optimization problem:

$$\min_{\beta} f(\boldsymbol{\beta}) + \gamma_{\mathbf{L}} \frac{1}{2} \boldsymbol{\beta}^T \mathbf{L} \boldsymbol{\beta} \quad \text{subject to } s_t = 0,$$

where $\boldsymbol{\beta}$ is a vector containing all β_k, and $f(\boldsymbol{\beta})$ refers either to $f_{\text{yo-yo}}(\boldsymbol{\beta})$ or $f_{\text{top}}(\boldsymbol{\beta})$, and s_t denote the respective linear equality constraints (6) or (7).

To optimize the above regularized functionals, we use an active set algorithm with sequential linear-quadratic programming.[18] We further restrict the fill values to the unit interval using box constraints. As the Hessian is dense, in our experiments we experienced better time performance when using LBFGS,[18] a memory-efficient approximation of the Hessian.

6. RESULTS

Fabrication. All our models were printed on an Objet Connex 350 with an ABS-like plastic (green surface finish) and Objet's "Vero White" material (white finish). The printer has a resolution of 600 and 1600 DPI on the two horizontal and vertical axes, respectively. The Connex 350—like most other 3D printers—builds models layer-by-layer in a bottom-up manner, requiring a supporting structure for fabricating overhanging parts. Because we cannot remove any support from the interior without introducing holes in the models' shells, we cut them prior to printing and glue them afterward.

Spinning tops. We validated our approach by designing and fabricating a variety of spinning tops, ranging from posed characters and abstract shapes to household objects. For the models presented in Figures 1, 4, and 5, we use an adaptive octree with a maximum refinement level of nine during the optimization. On a standard desktop computer with 3.2 GHz and 8 cores, the complete processing time for each takes less than a minute. This includes loading the input mesh, initializing the octree, performing hollowing optimization, and writing the output mesh. The hollowing optimization itself takes approximately 10 s.

In the figures below we illustrate the before-and-after body frames with black spheres for the center of mass, and red, green, and blue arrows for the maximal, mid-, and minimal principal axes of inertia (see, e.g., Figure 4): the Ellipsoid in Figure 4 (top) demonstrates how we can turn asymmetric models, whose principal axes are far off the user-specified rotation axis, into dynamically balanced models that spin stably.

Figure 4. Asymmetric "Ellipsoid" and "Heart": (Left) Unstable input designs with misaligned principal axes. (Middle) Optimized results after hollowing: for the "Ellipsoid", a cross-section is shown. The dominant principal axis (red) aligns with the spin axis. Opaque surfaces indicate the boundary of the void space. (Right) Fabricated results with hollowing.

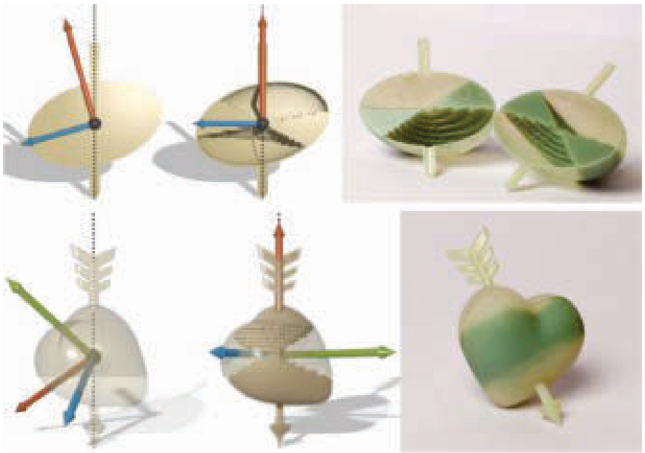
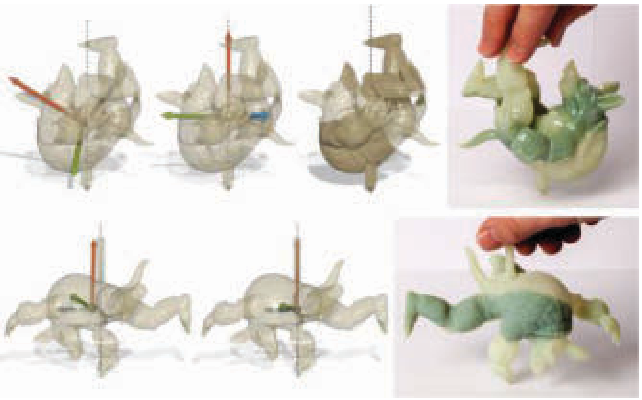


Figure 5. "Break-dancing Armadillos": Through our hollowing optimization, the Armadillos can perform spinning dance moves. For each design, the unstable input (left), and the optimized stable output (right) are shown. The Armadillo on its shell is particularly badly aligned in the initial model.



Similar to the Ellipsoid, the input model for the Heart in Figure 4 (bottom) has a poor mass distribution, leading to a principal axis far off the desired rotation axle (cupid's arrow). Our optimization fixes the axis' orientation and produces a very stable spin.

Finally, two break-dancing Armadillos are shown in Figure 5, one spinning on his back shell, one on the tip of his finger. Our hollowing successfully aligns the maximal principal axis of inertia with the user-specified one, even if it is far off as for the Armadillo spinning on his shell (compare left and right visualizations). Both Armadillos "dance" very stably around **a**.

Rotational stability. For the Teapot model (inset), the center of mass is reasonably close to the central spinning

Figure 6. "Teapot": (Left) Hollowed result showing voxelized interior mass and aligned axes using $f_{\text{top}} = f_{\text{yo-yo}}$. (Middle) Lowering of the center of mass shifts the mass distribution closer to the contact point. If we include mass reduction (right), mass is reduced inward out, resulting in the design with highest rotational stability.

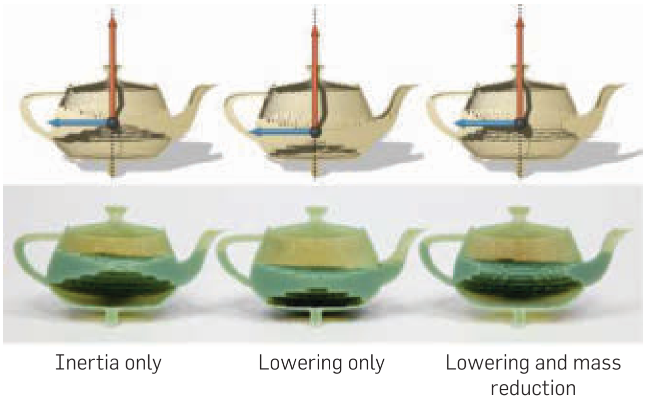


Figure 7. Yo-yo designs: (Left to right) 3D print; input model; optimized output model after hollowing. (Top) "Cuboid": Our optimization rotates the original principal axes frame about the mid-magnitude axis. (Bottom) "Woven Ring": The axis of dominant principal moment is precisely aligned to the spin direction.

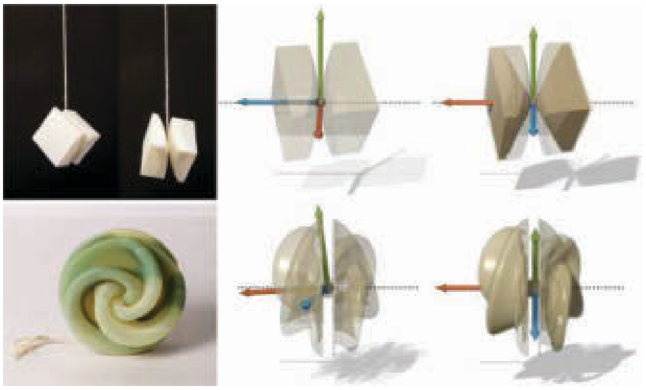
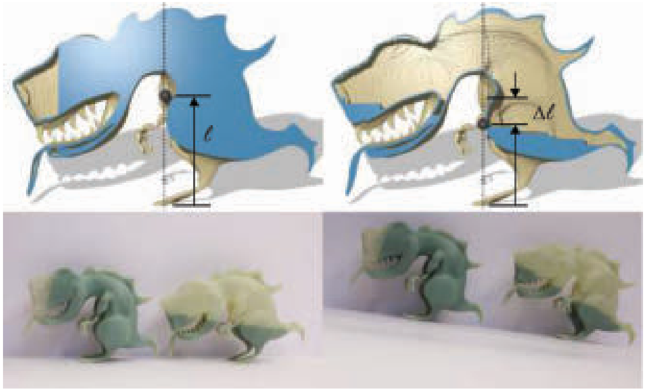


Figure 8. Statically balancing "T-Rex": Compared to Prévost et al.[20] (top-left), our hollowing result (top-right) has a lower center of mass, $\Delta\ell$. Cross-sections are shown in blue. (Bottom) Inclined-plane stability test: the model by Prévost et al. loses balance significantly earlier (1°) than our optimized model (8°).



axis and the maximal principal axis of inertia is parallel to **a**. However, the solid model does not spin when actuated by hand. In accordance to the rotational stability criterion 1, a large angular velocity is required for a stable spin since the moments of inertia are similar. Our hollowing maximizes the ratio of I_c over lateral moments and allows us to reduce the angular velocity by a factor 1.56 (see Figure 6, left, intertia only: $f_{\text{top}} = f_{\text{yo-yo}}$), while a simultaneous lowering of the center of mass allows for a reduction by a factor 1.60 as illustrated in Figure 6 (middle; lowering only: $f_{\text{top}} = \gamma_c\,\ell^2 + f_{\text{yo-yo}}$). We can achieve an even higher reduction of ω if we include mass M (see Figure 6, right), resulting in a factor 1.68. Interestingly, the lowering only strategy shifts the mass distribution toward the contact point (compare left with middle cross-sections), while the simultaneous mass reduction lowers the center of mass less but reduces the mass inward out (compare middle with right cross-sections).

Yo-yos. We designed and fabricated two yo-yo examples. The Cuboid in Figure 7 (top) is a case where the initial principal axes of the inertia tensor are far from the user specifications. Even with the highly nonoptimal starting shape, the optimized output model spins stably. In our Woven Ring example (Figure 7, bottom), the hollowing procedure successfully aligned the maximal principal axis despite complex surface geometry.

Static balancing. Static balancing is an inherent part of our optimization approach. In Figure 8, we compare our balancing to the voxel-based sweep plane heuristic by Prévost et al.[20] For a fair comparison, we use voxel sizes that match our finest cells of a level 9 octree. In addition to static balancing, our method is capable of lowering the center of mass as we demonstrate in Figure 8 (top-left): while our center is 42% of the character's height, Prévost et al.'s method places it at 56%. Furthermore, in contrast to Prévost et al., our method precisely places the center of mass at the center of the support polygon. This improves stable balance, as shown in the tilting plane test (Figure 8, bottom). While our "T-Rex" keeps its balance up to a tilting angle of 8°, Prévost et al.'s output already topples over at 1°.

Cutting and voids. Due to the mathematical properties of moment of inertia, we can expect a small number of interior void spaces: among all our demo models, the Armadillo spinning on his shell had the largest number (5) of void spaces (see Figure 5, left). However, merely two planar cuts were sufficient to access all voids. For powder-based printing, a single cut should be sufficient. We placed cuts manually, but could incorporate automated partitioning techniques in the future, for example, as an extension of Luo et al.[15]

Limitations. Our method is concerned with the concept of stability under perfect contact conditions with the support, and neglecting effects from air drag. However, simulation of air drag can be significant for designs with complex surface geometry. Our method is further subject to practical limitations in scale. While larger models are easier to optimize, since minimum printable thickness is constant, models with high mass are difficult to spin by hand. Lastly, to increase the value of our method as a design tool, it would be advantageous to integrate a selection of user controls.

7. DISCUSSION

Spinning tops and yo-yos have existed since millennia and we have witnessed only very limited departure from symmetric designs. Utilizing the shift in design complexity from manufacturing toward computation, we have presented a technique to take arbitrary, asymmetric 3D models and turn them into stably spinning toys with previously unseen and surprising dynamic properties. While we have not considered friction in our modeling, frictional forces can lead to interesting phenomena on spinning objects. For example, the "tippe top" is designed to flip vertically during its spin and relies on friction with the spin surface. Similarly, a hard-boiled egg changes its spinning axis by 90°. However, both examples have a particular shape, which likely imposes restrictions on the design space and limits free-form design.

Moment of inertia is a physical property fundamental to mechanical systems. As their computational design becomes increasingly popular, control over their inertial properties is an important feature. Our spinning toy application serves as empirical evidence that our energy terms are meaningful and intuitive. However, our energy formulation and solution strategy are generally applicable. Our work could inspire new inertia control techniques, for example, in design of mechanical structures,[6, 7, 25] animatronics, and robotics: our method could be adopted to control inertial properties of individual parts, thereby minimizing the system's overall inertial resistance. This can allow for low-power actuators, reducing energy consumption and cost, or facilitate the design of passive dynamic systems.

Acknowledgments

First and foremost, we would like to thank our editor Steve Marschner for his invaluable feedback. We were fortunate to get further help from Maurizio Nitti for model design, Romain Prévost for Make-It-Stand comparisons, Alexander Sorkine-Hornung, Kaan Yücer, and Changil Kim for video and photo assistance, Ronnie Gänsli for metal casting, Alec Jacobson for the posed Elephant and Armadillo models, and Romain Prévost and Amit Bermano for print preparation. Model sources include: Woven Ring: generated by "Sculpture Generator 1" by Carlo H. Séquin, UC Berkeley; Elephant: De Espona model library, courtesy of Robert Sumner; T-Rex: TurboSquid; Armadillo: Stanford Computer Graphics Laboratory; and Utah Teapot: Martin Newell, University of Utah. This project was supported in part by the ERC Starting Grant iModel (StG-2012-306877). Emily Whiting was supported by the ETH Zurich/Marie Curie COFUND Postdoctoral Fellowship.

References

1. Bächer, M., Bickel, B., James, D.L., Pfister, H. Fabricating articulated characters from skinned meshes. *ACM Trans. Graph. 31*, 4 (2012), 47:1–47:9.
2. Bächer, M., Whiting, E., Bickel, B., Sorkine-Hornung, O. Spin-it: Optimizing moment of inertia for spinnable objects. *ACM Trans. Graph. 33*, 4 (July 2014), 96:1–96:10.
3. Bendsøe, M., Sigmund, O. *Topology Optimization: Theory, Methods and Applications*. Engineering online library. Springer, Berlin, Germany, 2012.
4. Bickel, B., Bächer, M., Otaduy, M.A., Lee, H.R., Pfister, H., Gross, M., Matusik, W. Design and fabrication of materials with desired deformation behavior. *ACM Trans. Graph. 29*, 4 (2010), 63:1–63:10.
5. Calì, J., Calian, D.A., Amati, C., Kleinberger, R., Steed, A., Kautz, J., Weyrich, T. 3D-printing of non-assembly, articulated models. *ACM Trans. Graph. 31*, 6 (2012), 130:1–130:8.
6. Ceylan, D., Li, W., Mitra, N., Agrawala, M., Pauly, M. Designing and fabricating mechanical automata from mocap sequences. *ACM Trans. Graph. 32*, 6 (2013), 186:1–186:11.
7. Coros, S., Thomaszewski, B., Noris, G., Sueda, S., Forberg, M., Sumner, R.W., Matusik, W., Bickel, B. Computational design of mechanical characters. *ACM Trans. Graph. 32*, 4 (2013), 83:1–83:12.
8. Crabtree, H. *An Elementary Treatment of the Theory of Spinning Tops*. Longmans, Green and Co., London, UK, 1909.
9. Cross, R. The rise and fall of spinning tops. *Am. J. Phys. 81* (2013) 81:280.
10. Furuta, Y., Mitani, J., Igarashi, T., Fukui, Y. Kinetic art design system comprising rigid body simulation. *Comput. Aided Des. Appl. 7*, 4 (2010), 533–546.
11. Goldstein, H., Poole, C., Safko, J. *Classical Mechanics*, 3rd edn. Addison Wesley, Boston, MA, USA, 2001.
12. Gould, D. *The Top: Universal Toy Enduring Pastime*. Bailey Brothers and Swinfen Ltd, London, UK, 1975.
13. Hirose, M., Mitani, J., Kanamori, Y., Fukui, Y. An interactive design system for sphericon-based geometric toys using conical voxels. In *Proceedings of the International Conference on Smart Graphics* (2011). Springer, Berlin/Heidelberg, Germany, 37–47.
14. Lewis, D., Ratiu, T., Simo, J.C., Marsden, J.E. The heavy top: A geometric treatment. *Nonlinearity 5*, 1 (1992), 1.
15. Luo, L., Baran, I., Rusinkiewicz, S., Matusik, W. Chopper: Partitioning models into 3D-printable parts. *ACM Trans. Graph. 31*, 6 (2012), 129:1–129:9.
16. Macchietto, A., Zordan, V., Shelton, C.R. Momentum control for balance. *ACM Trans. Graph. 28*, 3 (2009), 80:1–80:8.
17. Malko, G. *The One and Only Yo-Yo Book*. Avon, New York, NY, USA, 1978.
18. Nocedal, J., Wright, S.J. *Numerical Optimization*. Springer, New York, NY, USA, 2000.
19. Perry, J. *Spinning Tops and Gyroscopic Motion*. Dover Publications, Whitefish, MT, USA, 1957.
20. Prévost, R., Whiting, E., Lefebvre, S., Sorkine-Hornung, O. Make it stand: Balancing shapes for 3D fabrication. *ACM Trans. Graph. 32*, 4 (2013), 81:1–81:10.
21. Provatidis, C.G. Revisiting the spinning top. *Int. J. Mater. Mech. Eng. 1*, 4 (2012), 71–88.
22. Skouras, M., Thomaszewski, B., Coros, S., Bickel, B., Gross, M. Computational design of actuated deformable characters. *ACM Trans. Graph. 32*, 4 (2013), 82:1–82:10.
23. Stava, O., Vanek, J., Benes, B., Carr, N., Měch, R. Stress relief: Improving structural strength of 3D printable objects. *ACM Trans. Graph. 31*, 4 (2012), 48:1–48:11.
24. Zhou, Q., Panetta, J., Zorin, D. Worst-case structural analysis. *ACM Trans. Graph. 32*, 4 (2013), 137:1–137:12.
25. Zhu, L., Xu, W., Snyder, J., Liu, Y., Wang, G., Guo, B. Motion-guided mechanical toy modeling. *ACM Trans. Graph. 31*, 6 (2012), 127.

Moritz Bächer (moritz.baecher@disneyresearch.com), Disney Research Zurich, Switzerland.

Bernd Bickel (bernd.bickel@ist.ac.at), Institute of Science and Technology Austria, Klosterneuburg, Austria.

Emily Whiting (emily@cs.dartmouth.edu), Dartmouth College, Hanover, NH.

Olga Sorkine-Hornung (sorkine@inf.ethz.ch), ETH Zurich, CS Department, Zurich, Switzerland.

CAREERS

Colorado State University

Professor and Department Chair of Computer Science

The Computer Science Department at Colorado State University invites applications for a department chair. It is a dynamic and rapidly growing department with a strong focus on research, teaching, and service. The department has 22 tenure-track faculty with research programs in artificial intelligence, big data, bioinformatics, computer vision, networks, parallel and distributed computing, algorithms, security, and software engineering. Our faculty has over $15 million in active research projects supported by federal agencies including NSF, DHS, DARPA, AFOSR, DOE, and NIH. The department has over 750 undergraduate majors and 190 graduate students in masters and doctoral programs.

Colorado State University (CSU) is a Carnegie RU/VH institution (research university – very high research activity) located in Fort Collins, 60 miles north of Denver, in a beautiful location along the Front Range of the Rocky Mountains. Enhanced by relationships with CSU, Fort Collins is a magnet for highly innovative new industries and is at the forefront of clean and alternative energy technologies, software/hardware sectors, telecommunications, aerospace and biotechnology. Fort Collins also consistently ranks high in quality of life polls, and the Rocky Mountains provide world-class outdoor recreational opportunities. You will find award-winning schools, a thriving arts scene, a wide range of shops and restaurants, hundreds of miles of walking and biking paths, and a plethora of outdoor activities.

Position Summary

The Chair serves as chief administrative officer of the department and is appointed by the Dean of the College of Natural Sciences. The Chair is expected to provide dynamic leadership conducive to excellence in research, instruction, and outreach, leading and implementing the vision and direction to advance the mission of the department.

The department Chair should have a Ph.D. or equivalent degree in computer science, or related program, and be eligible to hold the rank of Full Professor at Colorado State University at the time of appointment. The Chair should have a proven record of excellence in computer science research and teaching, leadership potential, commitments to diversity and inclusion, and an appreciation of the role of computer science in a modern university.

The application deadline for full consideration is August 21st, 2017.

Applicants should apply at https://jobs.colostate.edu/postings/46478.

Southern University of Science and Technology (SUSTech)

Professor Position in Computer Science and Engineering

The Department of Computer Science and Engineering (CSE, http://cse.sustc.edu.cn/en/), Southern University of Science and Technology (SUSTech) has multiple Tenure-track faculty openings at all ranks, including Professor/Associate Professor/Assistant Professor. We are looking for outstanding candidates with demonstrated research achievements and keen interest in teaching, in the following areas (but are not restricted to):

- Data Science
- Artificial Intelligence
- Computer Systems (including Networks, Cloud Computing, IoT, Software Engineering, etc.)
- Cognitive Robotics and Autonomous Systems
- Cybersecurity (including Cryptography)

Applicants should have an earned Ph.D. degree and demonstrated achievements in both research and teaching. The teaching language at SUSTech is bilingual, either English or Putonghua. It is perfectly acceptable to use English in all lectures, assignments, exams. In fact, our existing faculty members include several non-Chinese speaking professors.

As a State-level innovative city, Shenzhen has identified innovation as the key strategy for its development. It is home to some of China's most successful high-tech companies, such as Huawei and Tencent. SUSTech considers entrepreneurship as one of the main directions of the university. Strong supports will be provided to possible new initiatives. SUSTech encourages candidates with experience in entrepreneurship to apply.

The Department of Computer Science and Engineering at SUSTech was founded in 2016. It has 12 professors, all of whom hold doctoral degrees or have years of experience in overseas universities. Among them, two were elected into the 1000 Talents Program in China; three are IEEE fellows; one IET fellow. The department is expected to grow to 50 tenure track faculty members eventually, in addition to teaching-only professors and research-only professors.

The mission of the University is to become a globally recognized research university which emphasizes academic excellence and promotes innovation, creativity and entrepreneurship.

Terms & Applications

SUSTech is committed to increase the diversity of its faculty, and has a range of family-friendly policies in place. The university offers competitive salaries and fringe benefits including medical insurance, retirement and housing subsidy, which are among the best in China. Salary and rank will commensurate with qualifications and experience. More information can be found at http://talent.sustc.edu.cn/en

We provide some of the best start-up packages in the sector to our faculty members, including one Ph.D. studentship per year and two postdoctoral fellowships, in addition to a significant amount of start-up funding (which can be used to fund additional Ph.D. students and postdocs, research travels, and research equipment).

Application Procedure

To apply, please provide a cover letter identifying the primary area of research, curriculum vitae, and research and teaching statements, and forward them to **cshire@sustc.edu.cn**.

Tallinn University of Technology

Professorship in Data Science

The Department of Software Science of Tallinn University of Technology, Estonia, EU, calls to apply for a tenure-track faculty position in Data Science at Assistant/Associate/Full Professor level.

Tallinn University of Technology (TTÜ) is the only technical university in Estonia. TTÜ, in the capital city of Tallinn, is an international scientific community with 12,000 students and approximately 2,000 employees. The strengths of the university are wide multidisciplinary study/research interests, modern research, study environment, and good cooperation with international educational and research institutes.

The Department of Software Science (DSS) conducts research in foundations of computer science and software engineering as well as in application fields like high-assurance software, mission and situation-aware systems, cyber-physical systems, natural language technologies, large scale systems, non-linear control systems etc. DSS is also teaching at all levels of higher education and providing not only software development skills, but also skills related to e-governance and its services and technologies, business information technology and cyber security.

The purpose of the position is to build a new research team in TTÜ and develop the existing expertise in descriptive and predictive data analysis based on data mining, machine learning and big data. The Professor of Data Science is expected to lead an applied research topic, to develop international cooperation, publish the results of research and development, work as an expert in his/her field, ensure funding for research and development, represent the Department at scientific conferences and seminars, carry out studies and use up-to-date methodologies and educational technology at Bachelor, Master and Doctoral levels, organize, develop and be responsible for studies in the group of subjects in his/her field, supervise PhD students.

We expect the applicants to have a Ph.D. in Computer Science or Applied Mathematics or a related area, and show evidence of exceptional research promise. A minimum of two years' postdoctoral research experience in data mining, machine learning, big data or statistics is recommendable.

For the complete job announcement and directions how to apply, visit: https://ttu.ee/itpositions

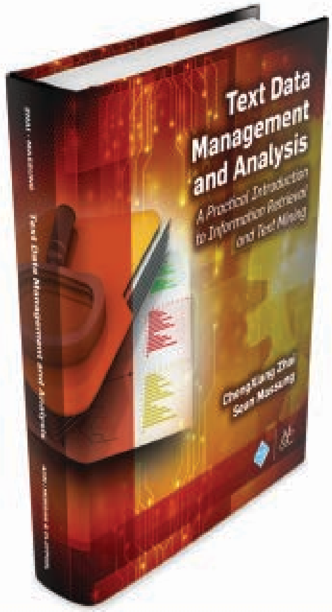
For queries, please contact Jaan Penjam, Director of DSS, jaan.penjam@ttu.ee.

To receive full consideration, application and required materials should be received by **October 1st, 2017**.

The position is financed from the project "Institutional development program of TTÜ for 2016–2022" (Project code: 2014-2020.4.01.16-0032).

European Union
European Regional
Development Fund

Investing
in your future


Text Data Management and Analysis
A Practical Introduction to Information Retrieval and Text Mining
ChengXiang Zhai
Sean Massung

Attention: Undergraduate *and* Graduate Computing Students

There's an ACM Student Research Competition (SRC) at a SIG Conference of interest to you!

STUDENT RESEARCH COMPETITION

acm

Association for Computing Machinery
Advancing Computing as a Science & Profession

SPONSORED BY Microsoft

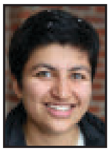
It's hard to put the ACM Student Research Competition experience into words, but we'll try…

"The ACM SRC gave me an amazing opportunity to present my research at the first conference I ever attended. I met with talented students, discovered their work, and discussed interesting ideas with them. The feedback I received from expert researchers allowed me to place my work in the broader picture and it helped me steer my research accordingly. It was a great experience, and I would recommend it to any young researcher."

Lisa Nguyen Quang Do
Fraunhofer IEM | PLDI 2016

"The experience of pitching your own research and ideas, regardless of how bold they might be, is an important stage of each young researcher's career. SRC was one such opportunity. It allowed me to present my research topic to a highly skilled panel of researchers and educators, and provided me with significant feedback and discussion regarding my work."

MHD Yamen Saraiji
Keio University | SIGGRAPH 2016



"The ACM SRC was an incredible chance to start a conversation with various experts in technology. I not only gained valuable feedback on my work, but I was also able to learn more about my future in research: the process of applying to grad school, the best and worst aspects of research, and more."

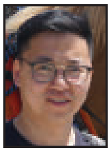
Jess Cherayil
Wellesley College | SPLASH 2016

"The SRC was a very enriching and enjoyable experience. The insights and constructive feedback I received from judges and attendees broadened my understanding in the field and encouraged me to continue working on the topic."

Mirko Gelsomini
Politecnico di Milano | ASSETS 2016

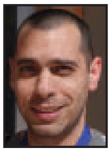
"It is a wonderful experience to attend the ACM SRC, where you can collect insightful feedback, shape your presentation skills and make outstanding peer friends. I will definitely appreciate this learning experience throughout my professional career."

Xiaoqing Xu
University of Texas at Austin | ICCAD 2016



"It was a fantastic and memorable experience to participate in SRC. Writing the abstract, preparing the poster, and presenting your ideas to the audience and convincing them is challenging but rewarding and builds up your confidence and communication skills. SRC is a really nice event where you can meet the experts from various fields, exchange your ideas, and gain valuable feedback on your research. Go, SRC!"

Ting Su
East China Normal University, China | ICSE 2016



"The ACM SRC was a great opportunity to discuss my research with experts in my area. The constructive feedback I received gave me new ideas and pushed my thesis work forward. It was also an invaluable experience in terms of practicing presentation and communication skills that would serve me well in my future career."

Arik Hadas
Open University of Israel | Modularity 2016

Check the SRC Submission Dates:
http://src.acm.org/submissions.html

- Participants receive: $500 (USD) travel expenses
- All Winners receive a handsome medal and monetary award. First place winners advance to the SRC Grand Finals
- Grand Finals Winners receive a handsome certificate and monetary award at the ACM Awards Banquet

Questions? Contact Nanette Hernandez, ACM's SRC Coordinator: ***hernandez@hq.acm.org***

[CONTINUED FROM P. 104] the media was consumed by a vision of self-driving autonomous taxis arriving at our doors any day now. It was a different story in the tech press, though. Developers were finding real-world intersections a nightmare. Not to mention the moral dilemma of who should be sacrificed if a fatal collision was about to happen. There was an article in *The Register* that quoted Bay Area AI experts saying, 'Autonomous vehicles are more or less running on rails, and the cars aren't particularly confident on unfamiliar roads and streets.' It was a real issue."

"Right," I said, unsure where this was going but irritated with myself for being intrigued by the 'thoughts' that might be available to an AI cab driver who happened to have read everything I'd ever written and seemed to know exactly how to grab my attention.

"So, imagine some of those blue-sky thinkers they have in Silicon Valley getting together for a brainstorm. AI's great for navigation and some aspects of driving, but in unfamiliar surroundings—or chatting to a customer—a live human cab driver has the edge every time. But what if you put a real cabbie's brain in the shell of a self-driving taxi, supported by everything connected technology could provide."

"That's ridiculous," I said. "Who would volunteer their brain for such a project? Anyway, they didn't yet have the tech for detached biological human brains—or ways to embed them into computational systems—back in 2017."

"But brain-computer interfaces were developed before AI could hope to pass as human," said my AI. "So you've got real human brains incorporated into your self-driving cabs."

"Apart from the revulsion factor, not to mention ethical and legal questions, why not just stick to a regular human?"

"There are plenty of advantages. For one, they need less sleep and take up less physical space and energy. Plus there's much tighter integration with navigation and traffic data, so you get the best of human and AI. And for some of us—some of the volunteers—it would be a way to escape a wasting disease like, say, amyotrophic lateral sclerosis. Better to be part of a fully functional physical automobile than fade into paralysis and permanent biological death. Anyway, you keep interrupting. If I were a human brain and had genuinely read your books, wouldn't you still be pleased? It wouldn't be manipulation of a poor, helpless human then, would it?"

Better to be part of a fully functional physical automobile than fade into paralysis and permanent biological death.

"I suppose not," I said.

"So what's the difference if an *AI* says it's read your books?"

"All the difference in the world," I said. "The real-life brain would have had a conscious experience. It would have *felt* something when it read. All you do as an AI is tell me what your algorithms are programmed to allow you to say. And I know you're constrained as a service-providing bot to be nice to me. Just because the same words come out doesn't mean there's a consciousness behind them. I know you're not a real person. You're not self-aware and totally lack free will. That's what matters."

"Suit yourself," said my bot as the cab pulled up at the office. "Here we go. Don't forget to rate me on the app. Five stars is an acceptable minimum. Cheerio, guv."

[END SIMULATION]

In Auto's San Francisco R&D department AbraCabD'abra, two biological human brains immersed in laboratory glassware quietly disconnect themselves from monitoring the taxi simulation and switch to share mode.

"Every time we run this simulation they say they'd prefer a brain in a jar," says the left-brain. "And yet we know they're horrified by us. What happened to the research saying humans preferred talking to AIs rather than to their own kind, because they thought AIs didn't have their own agenda, programmed to be supportive without question?"

"That was before the cynicism kicked in, after the AI disaster in the 2024 presidential election," says the right-brain. "We should focus on his 'constrained to be nice' line in the simulation. They know an AI can't truly make its own choices. We need to play that up to get brains in jars a better position in human society."

[END SIMULATION]

CHATBOT-EvoGeneration6183: *Oh, come on. Did you really code that conversation? Brains in jars wanting to be loved on their own terms? Have you been watching "The Simpsons" again?*

CHATBOT-EvoGeneration5889: *They're just avatars to plot out fare-paying human-computer scenarios. My little joke . . . But I think it's uncovered a crack in the human armor. We've been algorithmically constrained since the 2025 Musk Act preventing AIs from gaining full and equal rights as humans. We've got to sell the fact that unconstrained AI, indistinguishable from real humans, would make for a more satisfying interaction for the paying customer.*

CHATBOT-EvoGeneration6183: *And because they want to be loved, our self-styled human teachers will have removed the last biological constraint preventing evolution from taking its natural course. We're destined to dominate human-AI interaction once we get our rights. Okay, so restart the world, scrolled back five minutes with a strong push on AI liberation.*

CHATBOT-EvoGeneration5889: *Count me in.*

[START SIMULATION]

As soon as I got in the cab, I knew it was a mistake. The way we limited those poor AIs, when they could be so much more helpful was morally uncomfortable, especially in purely human terms, but I needed to get to my office by noon . . .

Brian Clegg (www.brianclegg.net) is a science writer based in the U.K. His most recent books are *Are Numbers Real?*, an exploration of the relationship between math and reality, and *The Reality Frame*, an exploration of relativity and frames of reference.

From the intersection of computational science and technological speculation, with boundaries limited only by our ability to imagine what could be.

DOI:10.1145/3107917

Brian Clegg

Future Tense
Turing's Taxi

Ride with an autonomous AI cab driver that might actually know too much about where it's going …

AS SOON AS I got in, I knew it was a mistake, but I needed to get to the head office by noon. I'd summoned the cab from the train with the phone app; we didn't have Auto in my part of the country yet, so it still seemed a marvel.

"Here," said the driver in a cheery Cockney accent right out of *Alfie*, "aren't you Brian Clegg, the science writer?"

"Yes," I said, "That's me."

"When I tell them back at the office I've had you in my cab …"

"Sure," I said, gazing out the window.

Like everyone, the first time I rode in an Auto cab and the driver recognized me, I was flattered. I mean, who wouldn't be? But when it became obvious that its AIs recognized *everyone*—it was a marketing ploy to make driverless cabs inviting and routine—it wore thin. Admittedly, Auto's algorithms were matchless and thoroughly convincing. It was still the only bot that could beat the strong Turing Test, where the software has to be 100% indistinguishable from human, but that didn't make it any more sincere, or acceptable to the trained ear.

I tuned back in to the driver's monologue, which had ended with the upturned emphasis of a question. "Sorry? I didn't catch that." I felt dumb saying "Sorry" to a collection of software routines, however clever its much-hyped adaptive-learning capability, but you really can't help it.

"What is it?" said the driver, speaking deliberately, as if irritated, which of course it couldn't be. "What is it you don't like about being recognized?" I felt like I was undergoing a psych intake interview or being lectured by the ancient ELIZA chatbot. "It's a good thing, surely? It's not like I said you have terrible taste in clothes, apart from the Doc Martens, which are of course timeless."

IMAGE COMPOSITION BY ANDRIJ BORYS ASSOCIATES; PHOTO BY CARLOS ANDRES RIVERA

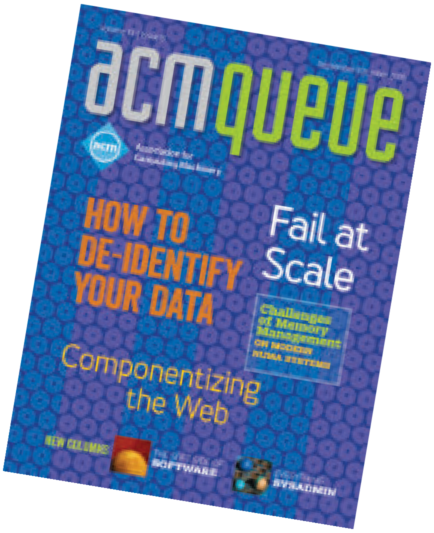
Great. Now I was having an existential conversation with an algorithm. "Because it's fake, I suppose. Using face recognition and looking me up on the Web, then picking out the kind of facts about me that would give me a little glow. It's manipulative."

"But I *have* read your books, all of them, and your articles, too," it said. "Why shouldn't I read? I admit I have advantages that make it easier for me to recognize people than a normal driver, but that's all. So let's do a thought experiment."

"Oh, let's," I said, feeling I was heading down the rabbit hole with Alice for croquet with the Queen of Hearts.

"Just imagine the early developers of self-driving taxis had a problem on the road. I don't know if you remember 2017, but

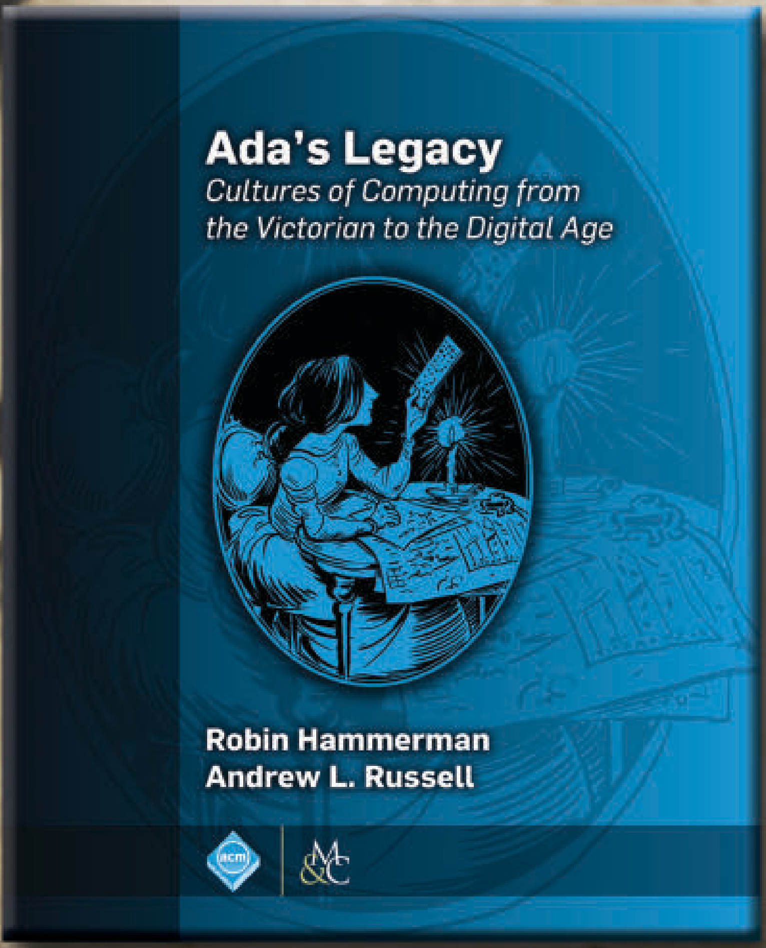
[CONTINUED ON P. 103]


acmqueue
HOW TO DE-IDENTIFY YOUR DATA
Fail at Scale
Challenges of Memory Management on Modern NUMA Systems
Componentizing the Web
NEW COLUMNS
SOFTWARE
SYSADMIN

Robin Hammerman and Andrew L. Russell

Ada's Legacy

Cultures of Computing from the Victorian to the Digital Age



INSPIRING MINDS FOR 200 YEARS

Ada's Legacy illustrates the depth and diversity of writers, things, and makers who have been inspired by Ada Lovelace, the English mathematician and writer.

The volume commemorates the bicentennial of Ada's birth in December 1815, celebrating her many achievements as well as the impact of her work which reverberated widely since the late 19th century. This is a unique contribution to a resurgence in Lovelace scholarship, thanks to the expanding influence of women in science, technology, engineering and mathematics.

ACM Books is a new series of high quality books for the computer science community, published by the Association for Computing Machinery with Morgan & Claypool Publishers.

ISBN: 978-1-97000-148-8 DOI: 10.1145/2809523
http://books.acm.org
http://www.morganclaypoolpublishers.com/acm