

DIGITAL ACCESS TO SCHOLARSHIP AT HARVARD
DASH.HARVARD.EDU

HARVARD LIBRARY


After Math: (Re)configuring Minds, Proof, and Computing in the Postwar United States

Citation

Dick, Stephanie Aleen. 2015. After Math: (Re)configuring Minds, Proof, and Computing in the Postwar United States. Doctoral dissertation, Harvard University, Graduate School of Arts & Sciences.

Link

http://nrs.harvard.edu/urn-3:HUL.InstRepos:14226096

Terms of use

This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material (LAA), as set forth at
https://harvardwiki.atlassian.net/wiki/external/NGY5NDE4ZjgzNTc5NDQzMGIzZWZhMGFlOWI2M2EwYTg

Accessibility

https://accessibility.huit.harvard.edu/digital-accessibility-policy

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. Submit a story

After Math

(Re)configuring Minds, Proof, and Computing in the Postwar United States

A dissertation presented

by

Stephanie Aleen Dick

to

The Department of the History of Science

in partial fulfillment of the requirements for the degree of Doctor of Philosophy

in the subject of the History of Science

Harvard University

Cambridge, Massachusetts

November 2014

© 2014 Stephanie Aleen Dick.

All rights reserved.

Dissertation Advisor: Professor Peter Galison Stephanie Aleen Dick

After Math

(Re)configuring Minds, Proof, and Computing in the Postwar United States

Abstract

This dissertation examines the history of three early computer programs that were designed to prove mathematical theorems: The Logic Theory Machine, the Program P, and the Automated Reasoning Assistant, all developed between 1955 and 1975. I use these programs as an opportunity to explore ways in which mathematical knowledge and practice were transformed by the introduction of modern computing. The prospect of automation generated disagreement about the character of human mathematical faculties like intuition, reasoning, and understanding and whether computers could be made to possess or participate in them. It also prompted novel discourse concerning the character of mathematical knowledge and how it should be produced. I track how the architects of each program built their beliefs about minds, computation, and proof into their theorem-proving programs and, in so doing, crafted new tools and techniques for the work of mathematics.

The practitioners featured in this dissertation were interested in whether or not computers could "think." I, on the other hand, am interested in how people think differently when they work with computers. And in particular how they thought differently *about mathematics* as they crafted a place for computers in the work of proof. I look for traces of their new ways of thinking in how they implemented their software. This is a new historiographical approach from existing history of computing that, for the most part, does not engage software at all or engages high-level descriptions or models of software.

I argue this: what were for my actors *implementation concerns* are in fact significant *epistemological issues* for the history of mathematics. Especially in the early decades, actually getting programs to run on computers was no small feat. In implementing programs practitioners had to craft many new tools, both formal and material - from programming languages and data structures to punched card encodings and user interfaces. The work of implementation spans multiple media - from paper to transistor - and involves many practices - from diagramming to coding - demonstrating that the media of the "digital age" are multiple indeed. Moreover, implementation is the site where we see practitioners rethinking their objects of interest, their disciplines, their theories, through the lens of computation - what is possible and impossible for computers to do. Implementation is the practice of automation.

In implementing their theorem-proving software, the actors in this dissertation gave new formulations and had new experiences of mathematical intuition, logical rules of inference, and other key tenets of twentieth-century notions of proof. The communities explored here were among the most influential and celebrated early contributors to automated theorem-proving. Each had quite a different relationship to the postwar American academic landscape relative to their disciplinary, institutional, and political makeup. Most importantly for they fundamentally and explicitly disagreed with each other about how the automation of proof should be done. Because of this, they developed very different automated theorem-proving programs – one seeking to simulate the human mind, another seeking to surpass it, and another seeking to develop theorem-proving software that would collaborate with a human user. Each of these projects intervened in the history of mathematics by introducing new forms – both social and technical – of proof.

Acknowledgements

My first debt of gratitude is to my advisor, Peter Galison. It would be impossible to thank him sufficiently here for his incisive reflections about my work and his unfailing advocacy and encouragement. I am also grateful to Peter for creating many opportunities for me to explore museum and theater projects related to my research. I share his belief that "making" and artistic collaboration can be truly enriching to academic work, and for me this was certainly the case. Peter invested in me both as a scholar and as a person and I relied on his brilliance, his kindness, and his guidance at every stage.

I also owe special thanks to Stefan Helmreich who was one of my most valued interlocutors and a tireless and amazing (and very close) reader of my work. Stefan's generosity and (lightning fast) insightfulness are unparalleled. My research was also deeply informed by a generals field I did with him in cultural and social anthropology. I can only hope that the dissertation adequately reflects the impact that anthropological theory, as wielded by Stefan, has had on my thinking.

I am extremely grateful as well to Jimena Canales, also a member of my committee, and also a gracious and close reader of my work. Jimena has an incredible knowledge of vast literatures and directed me toward many conversations that were important for this project but that I would not have discovered without her guidance. I also greatly valued her always honest, generous, and pragmatic advice about the dissertation itself and about the landscape of early academic careers.

I was honored to also have mathematician Barry Mazur on my committee. The late John Murdoch once said to me that Barry just "*sees things*," referring I think to the simultaneous breadth, depth, and seeming ease of his knowledge and understanding across many fields. I came to share this view, marveling at the subtlety and incisiveness of Barry's thinking in conversation, in seminars, and in his always thoughtful reading

of my work.

To Mario Biagioli and Joan Richards I also owe special thanks. Each supervised one of my generals fields and my thinking was shaped in significant ways through engagement with them. Both continue to be great mentors, readers, and colleagues.

I thank as well the other faculty in the Department of History of Science at Harvard, including especially Katy Park who was Director of Graduate Studies while I was a student as well as Janet Browne, Everett Mehndelson, and Charles Rosenberg for their advice, generous advocacy, and enthusiasm for graduate scholarship. I am very lucky to have had many other mentors whose seminars, conversations, collaborations, feedback, and interest were invaluable for my intellectual development - especially David Kaiser, Michael Gordin, Heidi Voskhul, Mark Schiefsky, Vincent Lepinay, Sophia Roosth, Sam Schweber, Nathan Ensmenger, and Amir Alexander. I am especially grateful to Nathan and Amir for affording me opportunities to publish pieces of this material. I must also thank Garth McCavana, Margot Gill and GSAS at Harvard for their unwavering support for graduate students of which I was a frequent beneficiary.

My graduate experience was, at every turn, shaped by the friendship, insights, and kindness of my fellow graduate students. My cohort was a singular example of colleagues who are also friends - I relied on them for advice, happy hour, feedback and, when needed, commiseration. The Modern Sciences Working Group, Phunday, and the Dissertation and Writers Group were central venues from which I received amazing feedback on many versions of this material and I am enormously grateful for them. Special thanks to Lukas Rieppel, Alex Czsisar, Alex Wellerstein, Dan Volmar, He Bian, James Bergman, Michael Rossi, Hallam Stevens, Henry Cowles, Ion Milhailescu, Colin Webster, Nassar Zakariya, and Evan Hepler-Smith for countless important conversations, wonderful feedback, collaborations, guidance, and many invaluable insights. I would also like to thank my cohort and mentors from the Harvard Horizons program for reflecting so thoughtfully on my work.

Among my graduate student colleagues, I must thank especially Meg Shields-Formato and Jenna Tonn. Both are dear friends and each of them read and thoughtfully engaged many pieces and versions of the material presented here. I have also, since the first day, truly looked up to Chris Phillips and Alma Steingart - I feel tremendously grateful to have them as like-minded colleagues, friends, and collaborators in the history of mathematics. *Vive la révolution!* Finally, I thank Myrna Perez Sheldon and Lisa Crystal who are among my best friends and closest colleagues and without whom

I would have been quite lost at many junctures.

I am grateful to the faculty and graduate students at the Institute for the History and Philosophy of Science and Technology at the University of Toronto where I completed a Master's degree, particularly Craig Fraiser and Chen-Pang Yeang. I also had the wonderful pleasure of visiting the Committee on Conceptual and Historical Studies of Science at the University of Chicago for the 2012 - 2013 academic year. They were incredibly generous hosts and I learned a great from their community and received excellent feedback upon presenting in the Fishbein Workshop in the History and Philosophy of Science.

I received other invaluable feedback while sharing this material at the Harvard Kennedy School STS Circle, the Mahindra Humanities Center, the Philadelphia Area Seminar on History of Mathematics, the European University at Saint Petersburg's *Explorers and Pirates* series, and several conferences including those of the History of Science Society, the Society for the History of Technology, the American Mathematical Society and the Mathematics Association of America, the International Congress of History of Science, Technology and Medicine, Computability in Europe, and the Harvard-Yale Graduate Conference in Book History.

This project would not have been possible without the incredible knowledge, skills, and generosity of many archivists. I thank the staff at the Rockefeller Archive Center, especially Bethany Antos for her *tour-de-force* processing of the Hao Wang Collection without which I would have been unable to access the collection in time for use here; Wendy Shay, Christine Windheuser, and Walter (Joe) Hursey from the Smithsonian Institute for their assistance with the John Clifford Shaw collection; Renu Berrett for help with the McMaster University Archives Bertrand Russell collection; Linda Staknke from the University of Illinois at Urbana Champagne archives; Roberta Shanman from the RAND Corporation; Dawn Stanford at the IBM Archives Department; the staff at the Carnegie Mellon University archives for help with the Newell, Simon, and McCorduck Collections; Amanda Pike from the Seeley G. Mudd Manuscrtip Library at Princeton University; the wonderful archivists and Debbie Douglas at the Massachusetts Institute of Technology.

This research was generously funded by a number of institutions, without which, the work would not have been possible. I currently hold a Turing Centenary Fellowship, funded by the John Templeton Foundation, as part of The Alan Turing Centenary Research Project: *Minds, Mechanism, and Mathematics*. I am extremely grateful to

Barry Cooper, the project PI, for his support. From 2008 - 2012 I was fortunate enough to hold a Doctoral Fellowship furnished by the Social Sciences and Humanities Research Council of Canada. I am also very grateful for support from the National Science Foundation, the Graduate School of Arts and Sciences at Harvard University, and the Hiebert Fund for the History of Science. During the final months of completing the dissertation, I was supported by the Harvard University Society of Fellows.

Finally, my family. My big sister Kristen has been looking after me since the playground days and is one of the most amazing women and greatest role models in my life. Her daughter Niah is a new and very loved friend who has brought me so much joy in the last year and a half. My little brother Travis has read more drafts of this material than anyone, with his brilliant and generous technical eye, and has been my closest collaborator, whether in games or academics, throughout my life. Most of all, my parents, Barry and Linda, who are two of my best friends. No one has worked harder to understand who I am and what I am trying to accomplish than they have. Thank you to each of them.

For my grandmothers, Nita and Elaine, who paved every way for me with their unwavering commitment to learning, teaching, working, imagining, and family.

Contents

Abstract iii

Acknowledgements v

List of Figures xii

Introduction: Reformalism 1

1 Rewriting *Principia*:
Implementing Intelligence 44

2 "Mathematical Objects in Action":
Implementing Herbrand's Theorem 96

3 A New Collaborator:
Implementing Intuition and Inference 153

Conclusion: Reflections on the Aftermath 193

Sources 212

List of Figures

1.1 Manuscript leaf from *Principia Mathematicia*. MU-BR, c.a. 1907. . . . 52

1.2 Typeset page from *Principia Mathematica*, first edition, 1910. 52

1.3 Logic Theory Machine Subproblem Proof Tree for Proposition 2.17 from *Principia Mathematica*. Newell, Shaw, Simon, "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristics," p. 228. . . 72

1.4 Tree Diagram of a Logical Expression. Newell, Shaw, "Programming the Logic Theory Machine," p. 10b. 73

1.5 Hand-drawn Logical Expression Trees by Simon. CMU-HS, RAND Series, "Graphic Materials", "Logic Theorist", not dated. 74

1.6 Hand-drawn Proof Trees by Simon. CMU-HS, RAND Series, "Graphic Material", "Logic Theory Machine." 75

1.7 Tree Diagram of *Principia Mathematica* Proposition 1.7: $\sim p \rightarrow (qvee-p)$. Newell, Simon, "The Logic Theory Machine," p. 9. 81

1.8 Diagram of a Linked List Information Structure. Newell, Shaw, "Programming the Logic Theory Machine", Rand Document P-954 (February 1957), p. 13a. 85

1.9 Diagram of Deleting an Element from a Linked List. Newell, Shaw, "Programming the Logic Theory Machine," p. 14a. 86

2.1 Tree Diagram of Program P Proof. Wang, "Toward Mechanical Mathematics," p. 6. 133

2.2 Hand Drawn Pattern Recognition Table from Wang's Notes. RAC-RU-HW, Series 3, Box 97, Folder 1369 - [Notebook, Bell Telephone Laboratories, 1959], p. 134. 147

4.1 CASC 2012 Results Display. My photograph. Manchester, UK, 2012. . 204

Introduction: Reformalism

Thinking (With) Machines

Alan Turing wanted to make an intelligent machine.[1] In 1948 he speculated that the 'surest way' would be to "take a man as a whole and to try to replace all the parts of him by machinery."[2] Turing imagined that this machine-man would become intelligent in the same way that its flesh and bone counterparts do, namely, by engaging with the world: "He would include television cameras, microphones, loudspeakers, wheels and 'handling servomechanisms' as well as some sort of 'electronic brain.' [...] In order that

[1]Turing (1912 - 1954) was a British mathematician and a pioneer of modern computing. In the 1930s, he developed some of the fundamental theoretical tenets of computation, and in the 1940s, he was instrumental in the development of some of the earliest modern computers. See Turing, "On Computable Numbers with an Application to the *Entsheidungsproblem*" in *Proceedings of the London Mathematical Society*, Vol. 42 (1937): 230 - 265. See also Charles Petzold, *The Annotated Turing: A Guided Tour Through Alan Turing's Historic Paper on Computability and the Turing Machine* (Indianapolis, IN: Wiley Publishing Inc., 2008). For a biography of Turing, see Andrew Hodges, *Alan Turing: The Enigma* (Princeton, NJ: Princeton University Press, 2012 [1983]).

[2]Alan Turing, "Intelligent Machinery," AMT/C, Unpublished Manuscripts and Notes, 11 *http://www.turingarchive.org/viewer/?id=127&title=1,* image 20. This essay was never published. Turing was on sabbatical at Cambridge University at the time he wrote the paper, having just notified the National Physical Laboratory (NPL) that he was leaving their employ for the University of Manchester. He was obligated to send copies of his work to Charles Galton Darwin, then head of the NPL. Darwin reported in an Executive Committee meeting that the essay "although not suitable for publication, demonstrated that [Turing] had been engaged in rather fundamental studies." (See "Minutes for the Executive Committee of the National Physical Laboratory" for 28 September 1948, http://www.alanturing.net/turing_archive/archive/l/l100/L100-001.html). According to Robin Gandy (as cited by Jack Copeland) Darwin also referred to the paper as a "school boy's essay" (Copeland, *Turing: Pioneer of the Information Age* (Oxford, U.K.: Oxford University Press, 2012): p. 190).

the machine should have a chance of finding things out for itself it should be allowed to roam the countryside."[3] This approach to making intelligent machinery was plagued with many difficulties, however. For one thing, if composed of late 1940s technology this machine-man would be of "immense size" and if allowed to roam the countryside "the danger to the ordinary citizen would be serious." Moreover, Turing observed that "the creature would still have no contact with food, sex, sport, and many other things of interest to the human being" and its ability to develop like a person would be limited. Ultimately, Turing concluded that "although this method is probably the 'sure' way of producing a thinking machine" it would be too slow and impractical. Instead, he proposed to "try and see what can be done with a 'brain' which is more or less without a body."[4]

The "electronic brains" that Turing had in mind were modern digital computers

[3] Alan Turing, "Intelligent Machinery," image 20 - 21.

[4] Alan Turing, "Intelligent Machinery," image 21. In this essay, Turing was agnostic about what precisely intelligence *is*. He neither defines intelligence nor suggests a method by which we might know if intelligence had been achieved in a machine or not. Instead he takes for granted that machines *could* become intelligent and explores different mechanisms by which this might be accomplished. Two years later he published one of his most influential papers, "On Computing Machinery and Intelligence," that took the exact opposite approach. In that later paper, he proposed a test for deciding whether or not a machine should be said to be thinking - a test that concerns only the behavior, specifically text-based and discursive behavior, of the machine and cares nothing for the mechanisms that produce this behavior. Also of note is Turing's emphasis on *embodiment* in relation to intelligence. He suggests in the unpublished 1948 paper that the production of intelligence is tied to a particular *form* - namely the human body - and to particular kinds of experience - roaming the world, sex, sport, food, and so on. "Embodied" rather than "embrained" conceptions of intelligence were not common in the 1940s. This suggestion of a relationship between body and intelligence is also different from Turing's 1950 paper in which he explicitly excludes bodies from his test for intelligence. Turing's test for intelligence was actually a variation of a parlor game concerning *gender*. In the original game, a man and a woman try to convince a judge of either sex that they are the woman (the man pretending, the woman as herself). The judge can ask them questions, answered on paper to hide their form, and must guess. Turing proposes that the computer be swapped for the man in this game, and it and a human try to convince the judge that they are the human, again with text-based discourse. Turing indicates that if the computer can fool the human judge that it is the person as often as the man can fool the judge that he is the woman, that machine should be said to think. One explanation for the original gender formulation of the Turing Test is that Turing was persecuted throughout his life for being gay. He was criminally prosecuted in 1952, imprisoned, and eventually chemically castrated, transforming his own body in several ways. The 1948 unpublished paper complicates the purely behaviorist and text-based "Turing Test" for which he became most famous by pointing to bodies, form, experience as vectors in intelligence. See Turing, "Computing Machinery and Intelligence" in *Mind*, Vol. 49 (1950): 433 - 460.

that he and others had been working to develop throughout the mid-1940s.[5] But what exactly could "electronic brains" do without bodies? What kind of intelligent behavior might they be capable of developing or exhibiting without being able to roam and engage the world? Luckily, Turing believed that there were a few domains of human intelligence that could be attained by a disembodied brain alone, because "they require little contact with the outside world."[6] Mathematics, he proposed, was one such domain.[7]

[5]The vocabulary of "electronic brains" was common among early descriptions of computers. See, for example, C. Dianne Martin, "ENIAC: Press Conference That Shook the World" in *IEEE Technology and Society Magazine*, (Winter 1995/1996): 3 - 10; Bernadette Longo, "Metaphors, Robots, and the Transfer of Computers to Civilian Life" in *Comparative Technology Transfer and Society* Vol. 5, No. 3 (December 2007): 253 - 273; Ted Friedman, "Filming the "Electronic Brain"" in *Electric Dreams* (New York, NY: NYU Press, 2005): 47 - 78. The computers that this language was crafted to describe emerged in the mid-1940s, though historians of computing hesitate to identify any definitive "first" such machine. A discussion of the contentious discussion surrounding "firsts" in computing, see Thomas Haigh, Mark Priestley, Crispin Rope, "Reconsidering the Stored-Program Concept" in *IEEE Annals of the History of Computing*, Vol. 36, No. 1 (2014): 4 - 17. General histories of early computing machines include Paul Ceruzzi, *A History of Modern Computing* (Cambridge, MA: The MIT Press, 2003 [1998]); Martin Campbell-Kelly, *Computer: A History of the Information Machine* (New York, NY: Basic Books, 1997); Larry Owens, "Vannevar Bush and the differential analyzer: The Text and Context of an Early Computer" in *Technology and Culture*, Vol. 27, No. 1 (1986): 63 - 95; Raúl Rojas, Ulf Hashagen, *The First Computers; History and Architectures* (Cambridge, MA: The MIT Press, 2000).

[6]Alan Turing, "Intelligent Machinery," image 21.

[7]He also included game playing and language translation in this category. These three domains became central focuses for Artificial Intelligence research throughout the second half of the twentieth century. There are different accounts of why AI practitioners converged upon these fields. For example, Alison Adam argues in *Artificial Knowing: Gender and the Thinking Machine* (New York, NY: Routledge, 1998) that each of these domains was thought to epitomize "intelligence" because they were male-dominated fields traditionally thought to epitomize "rationality," and "reason." These traits were historically gendered male and opposed to feminized "emotion" and "sensitivity" in schemes where women were associated with the body and men were associated with the mind. Harry Collins has argued, in *Artificial Experts: Social Knowledge and Intelligent Machine* (Cambridge, MA: The MIT Press, 1992) that because machines are not social beings, much of human knowledge will be inaccessible to them as knowledge is inherently social in character. According to Collins, machines will only ever be capable of participating in knowledge-making in domains where human activity in them has been mechanized and rule bound already in advance. Game playing and mathematics were such domains and that is what made them viable choices for Artificial Intelligence research. Pamela McCorduck offers an actor-informed narrative account of the emergence of Artificial Intelligence and the early choices of domains in *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence* (New York, NY: A K Peters, 2004). Early AI drew from an longer older historical tradition in which mind and reasoning processes - especially in mathematics - have been theorized, modeled, and understood in terms of machines and mechanisms. Examples of literature that explore the history of mechanism and mind include: Phil Husbands, Owen Holland, Michael

Early computers were used primarily for *calculation* - they were both functionally intended and conceptually understood as numerical data processing machines that could add, divide, solve differential equations, and so on.[8] This is not what Turing had in mind. Turing wanted *intelligent* behavior from these electronic brains, and calculation had long been stripped of that distinction and relegated to the domain of the merely "mechanical."[9] Turing hoped computers could operate in *nonnumeric* domains of mathematics - in logic, geometry, set theory, algebra, and so on in which problems and their solutions were not numerical.

Over the next two decades, it seemed that Turing's vision had been realized - beginning in the mid-1950s computers were hailed by some as "artificial mathematicians," "mathematical colleagues," and "mathematical assistants." These computers had been programmed to participate in one of the central activities of mathematical research - they were producing *proofs*.

As conceived in the twentieth century, proofs are deductive arguments that certain consequences must follow from certain premises. Proofs are meant to show that those

Wheeler, *The Mechanical Mind in History* (Cambridge, MA: The MIT Press, 2008); Roberto Cordeschi, *The Discovery of the Artificial: Behavior, Mind and Machines Before and Beyond Cybernetics* (Berlin: Springer, 2002); Margaret Boden, *Mind as Machine: A History of Cognitive Science, Volume 1&2* (Oxford, UK: Oxford University Press, 2002).

[8]Especially in the United States, early computing machines were used for the calculation of ballistics trajectories, numerical simulations of weather systems, nuclear weapons explosions and so on. See for example Harry Polachek, "Before the ENIAC" in *IEEE Annals of the History of Computing*, Vol. 19, No. 2 (1997): 25 - 30; Peter Galison, "Monte Carlo Simulations: Artificial Reality" in *Image and Logic: A Material Culture of Microphysics* (Chicago, IL: University of Chicago Press, 1997): 689 - 780; Thomas Haigh, Mark Priestley, Crispin Rope, "Betting on ENIAC: Nuclear Monte Carlo Simulations, 1947-50) in *IEEE Annals of the History of Computing* (forthcoming); Paul Edwards, *A Vast Machine: Computer Models, Climate Data, and the Politics of Global Warming* (Cambridge, MA: The MIT Press, 2013); Kristine Harper, *Weather by the Numbers: The Genesis of Modern Meteorology* (Cambridge, MA: The MIT Press, 2012). The transition from numeric computing to nonnumeric computing will be discussed further in the section "The Possibilities of Computing" below.

[9]Lorraine Daston has argued that at one time the ability to perform complex calculations was associated with a kind of virtuoso cognitive ability - it was held up as a hallmark of intelligence. However, once calculating machines were developed that could perform the same tasks faster and more effectively, calculation became identified as rote, mechanical, and base rather than exceptional, and remarkable in people. See Daston, "Enlightenment Calculations" in *Critical Inquiry*, Vol. 21, No. 1 (Autumn, 1994): 182 - 202.

consequences are true, that they are *theorems*. Proofs and the logical infrastructure thought to support them have often been thought to epitomize the powers of deductive reasoning and with it, a significant facet of human intelligence. If computers were proving theorems, then these so-called "electronic brains without bodies" would have made their way successfully into a domain long reserved exclusively for human thinking and human intelligence.

But were these computer programs really proving theorems? Were they actually intelligent? What exactly is a *proof*? Computers made possible new and powerful methods of theorem-proving but they also raised some fundamental questions. Some so-called computer proofs are too long for any person to read all the way through. Others make use of operations too complex for the human mind to easily understand and follow. Not everyone agreed that these should count as proofs. Not everyone agreed that computers were thinking. Not everyone agreed that thinking was required for theorem-proving.

These and related questions have been debated by mathematicians, philosophers of mathematics, and computing researchers since the advent of automated theorem-proving research in the mid-1950s.[10] The possibility of automating proof motivated

[10]The most infamous so-called "computer proof," and the one that motivated much of this disagreement, was that of the Four Color Theorem. The Four Color Conjecture asked a deceptively simple question: can every map be colored such that no two neighboring countries share the same color (with some caveats, e.g. all countries must be contiguous, barring disconnected pieces like Alaska)? The problem remained open for more than a century, in spite of sustained efforts on the part of numerous mathematicians. A proof was announced in 1976, by Kenneth Appel and Wolfgang Haken who enlisted the assistance of a computationally adept graduate student John Koch, and several computers in service of the proof. The proof was controversial for a number of reasons, but most especially because the computer contributions were not surveyable: no person could or has ever read through them all the way. As such, something other than a step by step reading of the proof would have to suffice to convince mathematicians that the result was correct - they had to trust the program, they had to trust the programmers, and they had to trust the computers. Readers interested in this proof and the surrounding controversy can consult Kenneth Appel, Wolfgang Haken, "Every Planar Map is Four Colorable" in *Illinois Journal of Mathematics*, Vol. 21, No. 3 (1977): 429 - 490; Appel, Haken, "The Four Color Proof Suffices" in *The Mathematical Intelligencer*, Vol. 8, No. 1 (1986): 10 - 20; Appel, "The Nature of Proof: Limits and Opportunities" in *The Two-Year College Mathematics Journal*, Vol. 12, No. 2 (1981): 118 - 119; Donald Albers, "Polite Applause for a Proof of One of the

practitioners to revisit the idea of "proof" itself, through the lens of computers and their perceived abilities and constraints. It also prompted novel and explicit debate about the character of human mathematical faculties like intuition, imagination, and reasoning, and whether or not computers could be made to possess them.[11]

This question - "can computers think" or how they might be made to do so - has been explored by many people.[12] That is not the question that motivates this dissertation. Instead I ask, *how do people think differently with computers?* And in particular how did they think differently *about mathematics* as they crafted a place for computers in the work of proof. This dissertation is a history of early automated theorem-proving programs. I am interested in how these programs were developed and in the new formal and material tools that were crafted in tandem. I ask - what happened to proof when these practices and tools, these programs, were introduced?

Mathematics is often held up as the most abstract of disciplines. Its truths are thought to obtain always and everywhere, its objects defy the confines of the physical world, and its deductive character is thought to escape the uncertainty of experiment

Great Conjectures of Mathematics: What is a Proof Today?" in *The Two-Year College Mathematics Journal*, Vol. 12, No. 2 (1981): 82; Donald MacKenzie, "Slaying the Kraken: The Sociohistory of a Mathematical Proof" in *Social Studies of Science*, Vol. 29, No. 1 (1999): 7 - 60; Stuart Shanker, "The Appel-Haken Solution of the Four-Colour Problem" in *Ludwig Wittgenstein: Critical Assessments* (New York, NY: Routledge, 2004): 395 - 412; Michael Detlefsen, Mark Luker, "The Four Color Theorem and Mathematical Proof" in *The Journal of Philosophy*, Vol. 77, No. 12 (1980): 803 - 820; Thomas Tymoczko, "The Four-Color Problem and Its Philosophical Significance" in *New Directions in the Philosophy of Mathematics* (Princeton, NJ: Princeton University Press, 1998): 243 - 266; Rudolf Fritsch, Gerda Fritsch, *The Four-Color Theorem: History, Topological Foundations, and Idea of Proof*, trans. Julie Peschke (Berlin: Springer, 1998).

[11]The field of automated theorem-proving is thus a "powerful disclosing agent" for beliefs about minds, computers, and proof. This phrasing comes from Lucy Suchman, who proposes that emotive robotics is a "powerful disclosing agent" for assumptions about the nature of human emotions. See Lucy Suchman, *Human–Machine Reconfigurations: Plans and Situated Actions (Learning in Doing: Social, Cognitive, and Computational Perspectives)* (Cambridge: Cambridge Univ. Press, 2006), p. 226.

[12]To name just a few, Alan Turing, "Computing Machinery and Intelligence" in *Mind*, Vol. 59 (1950): 433 - 460; Edward Feigenbaum, Julian Feldman eds., *Computers and Thought* (New York, NY: McGraw Hill, 1963); Harry Collins, *Artificial Experts: Social Knowledge and Intelligent Machines* (Cambridge, MA: The MIT Press, 1992); Hubert Dreyfus, *What Computers Still Can't Do: A Critique of Artificial Reason* (Cambridge, MA: The MIT Press, 1992).

and empiricism. Even Thomas Kuhn assigned a special status to mathematics, excluding it from his discrete model of revolution and rupture in the history of scientific knowledge.[13] I use automated theorem-proving as a way to suggest that even abstract objects have histories. I use it to explore the relationship between abstraction and materiality, between theorems and technology. I use it to emphasize that the way mathematics is *done* - by whom and with what - gives rise to particular abstract worlds, offers up particular questions and concerns, and ultimately shapes the character of mathematical knowledge and its objects.

It is a twentieth-century story. I trace how early work in automated theorem-proving produced different forms of proof than those of early twentieth-century mathematicians and logicians. In the early twentieth century, a particular culture of proof emerged that sought to reduce all of mathematics to logic - logicians wanted to formulate the branches of mathematics as deductive systems in which all truths could be produced by the application of clearly articulated inference rules to primitive principles, or axioms. This tradition of proof also had a particular material culture. It was a written, paper-based tradition - elaborate written symbol systems were designed to make the writing and reading of deductive proofs accessible and standardized. Proofs were collected in the pages of books meant to collect and consolidate mathematical knowledge in opposition to worry that its many branches were growing too far from one another through increasing specialization.[14]

Automated theorem-proving intervened in both the formal and the material dimensions of this early twentieth-century culture of proof. Early theorem-proving programs worked within formal deductive systems, often even proving the same theorems as those

[13]Thomas Kuhn, *The Structure of Scientific Revolutions*, third edition (Chicago: University of Chicago Press, 1996), p. 15.

[14]I discuss this tradition of proof at greater length in the section "Proving Theorems in the Twentieth Century" below, and in the section "A Mathematical World on Paper" in Chapter One.

penned by their human counterparts earlier in the century. But as we will see, they proved them differently. They made use of different rules of inference and different representation systems that were devised by their architects. They weren't proving theorems on paper but rather in the various electromagnetic media of early digital computation. When automated theorem-proving practitioners took elements of early twentieth-century proof and "put them into the computer," they changed them.[15] By the late twentieth century proof came in new and different forms.

To track this trajectory, I examine the development of three theorem-proving computer programs in particular: The Logic Theory Machine, developed by Herbert Simon, Allen Newell, and John Clifford Shaw in the mid-1950s at the RAND Corporation in Santa Monica, CA; the Program P, developed in the late 1950s by Hao Wang at Bell Systems Laboratories and later at the IBM Research Laboratory in Poughkeepsie, NY; and the Automated Reasoning Assistant, or AURA, developed in the early 1970s by Larry Wos, Stephen Winker, Ross Overbeek, John Alan Robinson, and others in the Applied Mathematics Division at the Argonne National Laboratory near Chicago, IL.[16]

I chose these three programs because they were among the most influential and celebrated early contributions to automated theorem-proving. I also selected them because, as we will see, their architects had quite different relationships to the postwar American academic landscape relative to their disciplinary, institutional, and political

[15]This phrase comes from Michael Mahoney, "What Makes the History of Software Hard," in *IEEE Annals of the History of Computing*, Vol. 30, No. 3 (2008): 8 - 18, p. 9.

[16]For published descriptions of these programs, see: Allen Newell, Herbert Simon, John Clifford Shaw "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristics" in *Proceedings of the Joint Western Computer Conference* (1957): 218 - 230; Hao Wang, "Toward Mechanical Mathematics" in *IBM Journal of Research and Development,* Vol. 4, No. 1 (1960): 2 - 22; Wang, "Proving Theorems by Pattern Recognition I" in *Communications of the ACM* Vol. 3, No. 4 (1960): 220 - 234; Larry Wos, "Solving Open Questions with an Automated Theorem-Proving Program" in *Proceedings of the Conference on Automated Deduction* (1982): 1 - 31; Larry Wos; Stephen Winker, "Open Questions Solved with the Assistance of AURA" in *Automated Theorem-Proving After 25 Years*, Vol. 29 of *Contemporary Mathematics*, eds. Woody Bledsoe, Donald Loveland (Providence, RI: American Mathematical Society, 1984): 73 - 88.

makeup. But primarily, I selected them because they were developed by practitioners who fundamentally and explicitly disagreed with each other about how the automation of proof should be done. Because of this, they designed and implemented very different automated theorem-proving programs which in turn introduced different forms of proof. These three programs came to serve as exemplars of the different approaches to theorem-proving adopted throughout the twentieth century.

The dissertation tracks their chronological development and the overarching debate in which each was situated. That debate concerned the central question of whether computers could or should prove theorems in the same way that their human counterparts do. The Logic Theory Machine was intended to simulate a human mind - its architects, Newell, Shaw, and Simon believed that at least in principle, the computer was capable of acquiring any faculties employed by human mathematicians when they proved theorems.

Wang disagreed. He believed that computers neither could nor should be made to simulate human reasoning practices, but instead their real contribution would be in performing mathematical analysis that was beyond the capacities of their human counterparts. The Program P was therefore designed to follow paths that the human mind was *incapable* of pursuing, and possibly that people would be incapable of even *reading*.

The Argonne team disagreed with both approaches, believing that computers would not be able to prove theorems on their own at all. Instead, they thought that significant results in mathematics would always require the direction and guidance of human insights and efforts. However, they also thought that there was room to capitalize on the power of computation to enlarge the world of mathematical exploration. As such, they designed theorem proving programs that could collaborate in real time with a human user to prove theorems. They thought they would capitalize on computers' and

humans' unique and different mathematical talents.

Each of these theorem-proving programs embodied different answers to questions about the character of human mathematical faculties, about the possibilities and limitations of computers, and about what kind of knowledge mathematical knowledge should be. These communities of automated theorem-proving practitioners operated with different beliefs about minds, computers, and proof - they were reconfigured in tandem.[17] Because of their different beliefs, these practitioners also developed very different theorem-proving programs. It is there, in the translation of belief and motivation into running program, that I believe some of the most interesting transformations in the character of proof took place.

I mean to invoke three distinct meanings of "after math" in the title of this dissertation. First, the dissertation itself is "after math" in the sense of being *in pursuit* of it. I want to know about mathematics - to understand its social, material, and historical character. I am following mathematics into a particular historical moment - one in which computers have become a new technological and conceptual resource for the work of proof. I follow mathematics into the media of digital computing, the sites where computing takes place, and into the hands of computing practitioners to see what became of it there.[18]

[17]I borrow this language of "reconfiguration" from Lucy Suchman, *Human-Machine Reconfigurations: Plans and Situated Actions* (Cambridge, UK: Cambridge University Press, 2007). I describe how my project corresponds to hers in more depth in the section "(Re)configuring Minds and Computation" in the Conclusion of the dissertation.

[18]I use this phrase "computing practitioner" to refer to all manner of people who worked on the design, operation, and programming of computing machines. I do not call them "computer scientists" since its eponymous discipline did not exist until the 1970s and the term also excludes certain people that I seek to include in my story, including engineers and operators of various kinds who were instrumental to the history of computing. For discussion of the various kinds of people and communities involved in early computing history, see John Alan Grier, *When Computers Were Human* (Princeton, NJ: Princeton University Press, 2005); Jennifer Light, "When Computers Were Women" in *Technology and Culture*, Vol. 40, No. 3 (1999): 455 - 483.; Nathan Ensmenger, *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise*; Thomas Haigh, "Inventing Information Systems: The Systems Men and the Computer, 1950 - 1968) in *The Business History Review*, Vol. 75, No. 1 (Spring 2001): 15 - 61

In another regard, the three computer programs on which the dissertation focuses are "after math" in the sense of being *in its image*. Each of the three computer programs were built to resemble mathematics - to embody ideas about what mathematics is, how it works, and how new knowledge is made within it. My historical actors were also "after math" when they encoded and translated it into new material forms.

Finally, in certain significant ways, we are living *after math*, in the sense of being "post" mathematics. Long held beliefs about what mathematics is, how we know in mathematics, how the work of mathematics is done, by who or what, have been challenged and continue to be challenged in light of the advent of computing. Mathematics has been opened up to include elements of other disciplines, like engineering, other ways of knowing, like experimentation and empirical exploration, and to new agents, like computers. Mathematics has changed, and in the aftermath of computing traditional foundations and epistemologies no longer suffice to define the work of proof.[19]

The Argument

I argue this: what were for my actors *implementation concerns* are in fact significant *epistemological issues* for the history of mathematics. There is no automation without invention, nor is automation merely an exercise in mere representation. Especially in the early decades, actually getting programs to run on computers was no small feat.

[19]In this regard, I follow Marilyn Strathern who in *After Nature* pursues a similar three-part exploration of the concept of "nature." Strathern is *after nature* in the sense of seeking to understand that word and its associated concepts, practices, and beliefs. She is also interested in perceptions of the "social" as being built "after the image" of nature which was long thought to provide the irrefutable facts of reproduction and kinship upon which culture was built. She is also interested in how the concept of "nature" has been troubled and dismantled in the face of assisted-reproductive technologies that throw into question the immutability of reproduction and kinship, long taken to the be the bedrock of natural facts to support cultural development. In this sense, she believes we are "after nature" in the sense of living in a world "post" irrefutable kinship infrastructure. Strathern, *After Nature: English Kinship in the Late Twentieth Century* (Cambridge, UK: Cambridge University Press, 1992).

In implementing programs practitioners had to craft many new tools, both formal - like programming languages - and material - like punched card encodings. The work of implementation spans multiple media - from paper to transistor - and involves many practices - from diagramming to coding - that constituted early "programming." Implementation is the site where we see practitioners rethinking their objects of interest, their disciplines, their theories, through the lens of computation.

My argument runs counter to existing accounts for example from, Jon Agar, that suggest that computing was not epistemologically significant in the early decades because it was introduced only to replace existing paper-based practices, in "contexts of pre-existing material practices of computation" (e.g. on paper, or with tabulating machines).[20] Early scientific computing was aimed at the automation of existing techniques and it was not until later that practitioners began to imagine radially new possibilities for knowledge-production with computing.

Agar was writing to temper enthusiastic claims of "revolution" in the sciences afforded by the advent of digital computing, and we should heed that analysis. I argue not for "revolution" in which the advent of computers caused sudden and dramatically different forms of knowledge-production. Rather I argue that communities had to overcome significant obstacles in making computers useful and usable, even as surrogate agents for existing practices, and that we should attend to those efforts when situating computers within historical epistemology.

I argue that existing paper-based techniques, or actually any existing techniques, can't be automated without the development of epistemically novel tools, the adoption of new perspectives, and the fashioning of new practices. But in order to recognize this, you have to look at *implementation*, a site usually neglected by historians of computing

[20] Jon Agar, "What Difference Did Computers Make?" in *Social Studies of Science*, Vol. 36, No. 6 (2006): 869 - 907, p. 899.

who tend to focus on high-level descriptions of computer programs or models rather than the practices that actually get them going. So this dissertation is in part meant to make a historiographic intervention - I advocate for *implementation* as a site that we should attend to when asking after the historical significance of computing.

The epistemological significance of implementation of the three programs I study here is related, in part, to the "displacement" of human mathematical agency involved in each. In making programs that would run on computers, the communities that developed the Logic Theory Machine, the Program P, and the AURA crafted new material and formal tools for the work of proof. And in every case, these were tools that *people could not use* because of their complexity, their size, or their structure. They were not tools with which people proved theorems, they were tools that enabled computers to prove theorems. In developing theorem-proving software, practitioners endowed mathematical objects with computational properties - they were translated into dynamic, algorithmic, discrete things.[21] They were stored and manipulated in the digital media of computing and there became electromagnetic and invisible and inaccessible to the human eye and hand. Their development involved a step back, an indirect perspective in which control of the objects of mathematics and the practices of proof were assigned to quite different agent.

In addition to non-human oriented tool development, each program also produced new and surprising *results*. The Logic Theory Machine produced a previously unknown proof of a logical theorem. The Program P revealed that certain branches of logic had a shared structure, a result of interest to logicians but previously unknown. And the AURA-user team from Argonne produced solutions to open problems from branches of mathematics about which the users did not have extensive knowledge - it enabled them

[21]For an interesting discussion of the "dynamism" and processual behavior of things in digital computing, see Brian Cantwell Smith, *On the Origin of Objects* (Cambridge, MA: The MIT Press, 1998): e.g. 34 - 37.

to work in problem domains beyond their training. Each result surprised its developers and users. Each result represented a different *kind* of computer contribution, aligned with the approach to automation that motivated its design. And each result was produced with the non-human oriented toolkit described in its implementation. The computer programs produced new results and new insights about mathematics.[22]

In developing and using their programs, practitioners came to know different things about mathematics and know them differently. New ways of asking and answering questions about proof became possible with those tools in hand. These tools endowed existing mathematical systems and ideas with new and different meaning. As anthropologist Stefan Helmreich writes, "when abstractions are realized in particular media, the media make a difference to how the abstractions are understood."[23] In taking seriously the relevance of materiality for abstraction, I also follow Peter Galison who has explored the complex relationships between different material cultures and epistemic traditions in twentieth century physics.[24] To capture the process by which mathematics was given new formal and material properties through translation to computing, I have coined the term "reformalism" to accompany the existing term, "remediation" due to Jay Bolter and Richard Grusin.[25] The term is meant to capture the relationship of abstraction and materiality which transform in tandem in the hands of practitioners

[22]I discuss these results in the concluding section of each chapter, and also in the section "The Possibilities of Computing" below.

[23]Stefan Helmreich, *Sounding the Limits of Life: Essays in the Anthropology of Biology and Beyond* (Princeton, NJ: Princeton University Press, forthcoming)).

[24]Galison, *Image and Logic: A Material Culture of Microphysics* (Chicago, IL: University of Chicago Press, 1997); Galison, *Einstein's Clocks and Poincaré's Maps: Empires of Time* (New York, NY: Norton, 2004).

[25]See Jay Bolter, Richard Grusin, *Remediation: Understanding New Media* (Cambridge, MA: The MIT Press, 1999). This idea also resonates with the work of Jonathan Sterne, especially in *MP3: The Meaning of A Format* (Durham, NC: Duke University Press, 2012). in which he proposes that any "media theory" should be accompanied by "format theory" in which format "denotes a whole range of decisions that affect the look, feel, experience, and workings of a medium" and a "set of rules according to which a technology can operate" (p. 7). Format and form are not the same thing, and Sterne is not primarily interested in relationships of abstraction and materiality. However, we share an interest in understanding the complex networks of practices, ideas, and people that make new media *work*.

who work at their interface.

Each chapter in what follows explores how certain pieces of the world of mathematics were put "put into computers" and in the process took on aspects of their formal and material character. The abstract and the concrete, the material and the formal, the technological and the human can be seen, on the ground, to develop in and through one another. In Chapter One, "Rewriting *Principia*: Implementing Intelligence" I explore how the architects of the Logic Theory Machine developed a model of human theorem-proving practice as "heuristic search" through a branching tree of problems and related subproblems. In order to program that model, they devised a new representation system for logic - one based on what were called "linked list" information structures and list processing operations. In so doing, they introduced new structures, new properties, and new processes to logic and to the work of proof.

In Chapter Two, "Mathematical Objects in Action: Implementing Herbrand's Theorem" I explore how Hao Wang transformed an an existing logical theorem *about* logical proof into a set of tools for actually *doing* logical proof. Like the Logic Theory machine, the Program P was designed to prove theorems from *Principia* but it did so in a completely different way. Wang developed a set of what he called "pattern recognition" tools or the "method of sequential tables" to enable the computer to prove theorems. These new tools were inspired by an existing result from mathematical logic called Herbrand's Theorem, one of the fundamental early twentieth-century results from proof theory. The theorem establishes a particular abstract relationship between two branches of logic. In doing so, it provides an "in principle" method for deciding if a statement in the former is a theorem using statements from the latter. However, it would actually be impossible for a person to use Herbrand's theorem to actually prove theorems except in a very small number of cases given that it would involve the analysis of *huge* (trillions) numbers of cases, even for a simple example. Wang devised

a method for putting Herbrand's theorem to work in the form a computer program and in so doing gave a computer-oriented reformalization to Herbrand's theorem and introduced a new set of computational tools to proof theory.

Finally in Chapter Three, "A New Collaborator: Implementing Intuition and Inference," I track how two traditionally central tenets of mathematical practice - intuition and inference - are given new computer-oriented formulations in the design of a collaborative theorem-proving program. The Argonne team reserved "intuition" for the human user to provide, believing that it could never be automated and would always be required for important mathematics. However, in order to *input* intuitions to a computer, to make them useful for the program, they developed a "Weighting Mechanism" in which their intuitions were translated into quantitative templates. Even in cordoning off intuition as something uniquely human, they had to translate it into the vocabulary and technical affordances of computing in order to actualize their vision of collaboration. The Argonne team also developed a new inference rule, called the "Resolution Principle." Where traditional laws of inference were created to capture the basic and primitive units of *human* reasoning and cognition, Resolution was designed to capitalize on the computers' speed and efficiency. It was a machine-oriented rule. In each program, some component of mathematics - a written symbol system, an existing theorem, intuition, and inference - were translated so as to accommodate the affordances of computing, and in tandem, given new material and formal characters.

In Search of Software

Software in general, has only recently begun to attract the attention of historical research. Early histories of computing were instead focused on the machines themselves

or on handfuls of famous practitioners.[26] In large part following the prescriptions of historian Michael Mahoney, however, recent scholarship has turned increasingly to historical explorations of computing *communities*. Mahoney argued that, unlike technological artifacts that preceded it like the steam engine and printing press, "the computer" has no intrinsic character but that the behavior and significance of computing depends on how it is *used* by different communities. And those communities harness the computer for their ends by designing or using *software* that transforms the "protean machine" into a particular kind of tool.[27] They "put their portion of the world into the computer."[28] As such, according to Mahoney the history of computing should be a history of communities and their software, and I agree.

However, especially in the early decades of computing, the distinction between *hardware* and *software* - now usually associated with "material" and "abstract" respectively - was not well defined. As we will see, the abstract and the concrete, the hardware and the software, the formal and the material were not easily teased apart in the work of early computing practitioners. In designing and implementing automated theorem-proving programs, the practitioners I study here developed both abstract and concrete tools - they designed abstract algorithms and formal structures that were informed by the material limitations of their computers' storage devices. I focus on implementation because it stands at the intersection of the material and the abstract of computing - it is where ideas and technology meet, it is where formal abstraction becomes executable program - it is where the computer both *enables* and *constrains* its users.

Mahoney was right that different computing communities fashioned the computer

[26]Nathan Ensmenger offers an excellent survey of the historiography of history of computing in "The Digital Construction of Technology: Rethinking the History of Computers in Society" in *Technology and Culture*, Vol. 53 (2012): 753 - 776.

[27]Mahoney, "The Histories of Computing(s)" in *Interdisciplinary Science Reviews*, Vol. 30, No. 2 (2005): 119 - 135, p. 122.

[28]See Mahoney, "What Makes the History of Software Hard," p. 8.

into all kinds of different things, but he may go too far in writing that "whereas other technologies may be said to have a nature of their own and thus to exercise some agency in their design, the computer has no such nature."[29] It certainly is the case that computers were wielded differently in the hands of different communities by way of their programs, but it is also the case that those communities encountered at every turn the obtuse material limitations of their machines.[30] Encounters with those constraints can be clearly seen when practitioners address the question - so how do we actually make this run? How can this be done with this machine's limited storage and processing resources? Implementation is where we see practitioners accommodating the affordances of computing while harnessing it for their design.[31]

By emphasizing implementation, I diverge from existing historiography. Implementation has not been prioritized in histories that explore the significance of computers in knowledge-making. This may stem in part from the fact that computer scientists themselves tend to emphasize abstract algorithms and high-level descriptions of programs

[29]Mahoney, "The Histories of Computing(s)," p. 122.

[30]I think Mahoney also appreciated this fact. Elsewhere, he wrote the following: "The history of software is the history of how various communities of practitioners have put their portion of the world into the computer. That has meant translating their experience and understanding of the world into computational models, which in turn has meant creating new ways of thinking about the world computationally and devising new tools for expressing that thinking in the form of working programs," in "What makes the history of software hard" p. 8. I suspect Mahoney emphasized the social dimension - the significance of user communities for determining the character of computing - and the abstract dimension - the creation of new models and ways of thinking, over the the production of "working programs" on the ground, because he was writing *against* a number of technologically determinist accounts in which the computer came along and "impacted" the way people operated. I, of course, also do not subscribe to technological determinist accounts of the history of computing, but I do think it is important to investigate how the physical design of computers, the limitations of computing resources, the kinds of operations particular computers could perform efficiently, etc. as well.

[31]An excellent discussion of the "ephemerality" and software and the associated difficulty of identifying where and how to write its history can be found in Wendy Chun, "Introduction: Software, a Supersensible Sensible Thing" in *Programmed Visions: Software and Memory* (Cambridge, MA: The MIT Press, 2011): 1 - 11. A historical survey of early changing conceptions of "software" can be found in Thomas Haigh, "Software in the 1960s as Concept, Service, and Product" in *IEEE Annals of the History of Computing*, Vol. 24, No. 1 (2002): 5 - 13. Michael Mahoney also writes about the difficulty of writing the history of software in Mahoney, "What Makes the History of Software Hard."

over low level implementation details.[32]

Today, high-level models and algorithms might be implemented in any number of existing programming languages equipped with libraries of ready-made variables and functions. The increasing availability of implementation resources makes the separation of higher and lower level program development easier. But in the early decades of computing, implementation often involved the creation of computational tools from scratch - the high level algorithms and the programming languages, data structures, and punched card encodings, were created by the same people, at the same time, and in service of the same project. While I advocate for a historiographical attention to implementation throughout the history of computing, I argue that it is a particularly important concern for histories of the early decades of computing of which this dissertation is a study.

My emphasis on the implementation of software comes with a question about sources. Any history of software is complicated from the start by the slippery ontological character of computer programs themselves. What are they? Where do we look for them? Are programs reducible to their source code? Do they live inside the memory of the computers that store and run them? Do they live on the screen, in the interface crafted for meaningful use, feedback, and interaction? Are they constituted, at heart, by abstract algorithms or concrete technologies?

[32]One explanation for this tendency is historical. During the 1960s and 70s, when some computing practitioners argued for an academic discipline of "computer science," they highlighted the theoretical tenets of their work - the study of algorithms and of the limits of computation - over the more practical and low-level elements of program development. They argued that computation ought to be the subject of scientific inquiry for its own sake, rather than a tool used only in service of other domains. And more, they themselves didn't want to be seen as mere technicians but as scientist. The emphasis on higher-level elements of computation was related to this professionalization moment, and seems to have been adopted by historians as well. For a discussion on the emergence of computer science as an academic discipline, see Dick, "Computer Science" in *A Companion to the History of American Science* (Hoboken, NJ: Blackwell Publishing, forthcoming). See also the discussions in Nathan Ensmenger, *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise* (Cambridge, MA: The MIT Press, 2010); Michael Mahoney, *Histories of Computing*, ed. Thomas Haigh (Cambridge, MA: Harvard University Press, 2011).

The supposed dichotomy between "software" and its opposite "hardware," lends itself to a treatment of the former as immaterial, abstract, ephemeral. Software has been associated with the abstract, in part, because it is associated with *mathematics* - the study of abstract formalism, objects, and relations. Indeed, in its first deployment, the word "software" was used to designate the "mathematical and logical instructions for electronic calculators."[33] But just as some historians of mathematics are working to demonstrate that the abstract worlds of mathematics are always and everywhere tethered to material tools and techniques, so too have some media and computing historians sought to materialize software.[34]

By exploring implementation, I seek to open up software as a *practice.* This approach displaces the object, the "program" itself, and points instead towards the processes and tools that accompany their development and use. This is a story about the *work* that was done to make computers useful and usable for mathematical theorem-proving. By focusing on practice, rather than on programs themselves, several sites and materials become visible. Software, it turns out, is *multimedia.* From page to tran-

[33]Matthew Fuller, "Introduction," in *Software Studies: A Lexicon*, ed. M. Fuller (Cambridge, MA: The MIT Press, 2008): 1 - 13, p. 2.

[34]For example, Reviel Netz has shown how the ancient Greek tradition of deductive reasoning was tied to a particular set of writing and diagramming practices. See Netz, *The Shaping of Deduction in Greek Mathematics: A Study in Cognitive History* (Cambridge, UK: Cambridge University Press, 2003). Lorraine Daston has explored the relationship of physics and mechanics to a nineteenth century traditions of synthetic geometry in "The Physicalist Tradition in Early Nineteenth Century French Geometry" in *Studies in History and Philosophy of Science*, Vol. 17, No. 3 (1986): 269 - 295. Herbert Mehrtens has explored the changing meaning and use of mathematical models in "Mathematical Models" in *Models, The Third Dimension of Science*, ed. S. De Chaderevian, N. Hopwood (Stanford, CA: Stanford University Press, 2004): 276 - 304. One might also include Brian Rotman's semiotic work on mathematics as text in, for example *Mathematics As Sign; Writing, Imagining, Counting* (Stanford: Stanford University Press, 2000). On the materiality of software, see for example, Matthew Kirschenbaum who has explored "traces" left behind when digital information has been erased or destroyed in order to emphasize the material robustness of that information, so often imagined to be ephemeral and immaterial. Matthew Kirschenbaum, *Mechanisms: New Media and the Forensic Imagination* (Cambridge, MA: The MIT Press, 2008). See also, for example, historian of computing Michael Mahoney who alternatively emphasizes the dynamism of software: "In essence [software] is the behavior of the machines when running. It is what converts their architecture into action, and it is constructed with action in mind." Michael Mahoney, "The History of Computing in the History of Technology" in *IEEE Annals of the History of Computing*, Vol. 10, No. 2 (1988): 113 - 125, p. 121.

sistor, from diagram to data structure, the work of implementation mobilizes thought-experiments, metaphors, correspondence networks, thousands of pages, hand-drawn experiments, technical reports.

My account is based upon the traces my actors left behind while doing the work of implementation. They left a paper-based trail that reveals how and why they crafted each computational tool the way they did to actualize their visions of automated proof. My research was primarily archival - I work with hand-written notes in which my actors experimented on paper, looking for different structures and operations to include in their programs. I explore the technical reports in which they circulated descriptions of their program designs and justifications for particular design choices. I also study their "dictionaries" in which practitioners collected, named, and specified the various operations that constituted their programs, programming languages, and representation systems.

Where they existed, I also explore the reference guides and user's manuals that describe how programs or their associated programming languages can be used. I also rely heavily on the surrounding documentation in which automated theorem-proving practitioners corresponded with one another, trying to make sense of or improve the behavior of their programs. I also explore the extensive correspondence and publications in which aligned their work with other research programs, with particular disciplines, and with other fields of computing. This surrounding documentation often reveals why practitioners believed that a particular implementation best fulfilled their overall vision for a mathematics that included computing.

Given that my methodology was largely paper-based, I should note that I claim no direct access to the materiality that I propose was central for my actors' rethinking and retooling of mathematics. I do not work with their computing machines or the programs they ran. I am rather interested in evidence of *their* encounters with those

machines. I look for traces of that materiality in the remaining records of their work. I would not expect to have the same experiences as my actors were I to encounter the material artifacts themselves, but rather want to recover and reconstruct the specificity of their encounters through the archive.

Proving Theorems in the Twentieth century

I am interested in how early theorem-proving programs intervened in the history of mathematical proof. In particular, I explore how the developers of each program were both drawing from and transforming a particular tradition of proof that emerged in the early twentieth century. That tradition embodied a desire to reduce mathematics to logic: to construct the branches of mathematics as formal axiomatic systems. Proofs in this tradition were meant to have a particular form: they consisted of the application of deductive rules of inference to the axioms or primitive principles of a formal logical system.

In 1895 Giuseppe Peano, an Italian mathematician and early developer of mathematical logic, first published a work called *Formulaire de Mathematique*.[35] It was intended as a catalogue of all mathematical knowledge consisting, literally, of numbered lists of mathematical theorems. Peano's goal was to collect and circulate "all

[35]Peano, *Formulaire de Mathemati*que, (*Rivisita di matematica*, 1985). The text was published in five editions, each an elaboration and expansion of the previous. The fifth edition bears a different name - *Formulario Mathematico* - and was published in 1905. A helpful introduction to the work is available from Hubert Kennedy, *Peano: The Life and Works of Giuseppe Peano* (Berlin: Springer, 1980), esp. Chapter 6 "The *Formulario* Project", pp. 44 - 50 and Chapter 17 "Completion of the *Formulario*", pp. 118 - 124. Peano is perhaps best known for his contributions to the development of symbol systems and notations in mathematical logic. He is also well known for his development of an axiom system for the natural numbers. Histories of the emergence of mathematical logic in the late nineteenth century that discuss Peano's life and work is Ivor Grattan-Guinness, *The Search for Mathematical Roots, 1870 - 1940: Logics, Set Theories and the Foundations of Mathematics from Cantor through Russell to Gödel.* (Princeton, NJ: Princeton University Press, 2002), esp. pp. 219 - 267 and N. I. Styazhkin's traditional account of the history of logical "ideas and attitudes", *History of Mathematical Logic from Leibniz to Peano* (Cambridge, MA: The MIT Press, 1969).

established results from all branches of mathematics."[36] He wanted to standardize and organize mathematical knowledge in one place.

The work was motivated in part by a growing concern on the part of mathematicians, logicians, and philosophers of mathematics in the late nineteenth and early twentieth century that the mathematical knowledge they inherited may be inconsistent and rife with contradictions. These sentiments emerged in part because of the discovery of certain contradictions and paradoxes within mathematics, and especially set theory. It also emerged as mathematicians were becoming increasingly distributed among fields and subfields of research that often adhered to different standards and that often did not speak to another.[37]

[36] Alex Csiszar, "Poincaré and Peano: bibliographer vs. natural philosophy," [unpublished excerpt] p. 1.

[37] This period has often been called "The Foundations Crisis", following Herman Weyl's publication "Über die neue Grundlagenkrise der Mathematik" in *Mathematische Zeitschrift*, Vol. 10 (1921): 39 - 79. Bertrand Russell himself constructed a paradox within Georg Cantor's set theory. Andrew Irvine summarizes it as follows: "Some sets, such as the set of all teacups, are not members of themselves. Other sets, such as the set of all non-teacups, are members of themselves. Call the set of all sets that are not members of themselves "R." If R is a member of itself, then by definition it must not be a member of itself. Similarly, if R is not a member of itself, then by definition it must be a member of itself." See Irvine, "Russell's Paradox" in *Stanford Encyclopedia of Philosophy,* ed. E. Zalta (Summer 2009): http://plato.stanford.edu/archives/sum2009/entries/russell-paradox/) on p. 1. That such contradictory cases were constructable within Cantor's set theory and elsewhere in mathematics was worrisome to the community. Large scale efforts to find solid grounding for mathematics were pursued in continental Europe, England, and the emerging mathematical community in the United States. See Ivor Grattan-Guinness, *The Search for Mathematical Roots, 1870 - 1940: Logics, Set Theories and the Foundations of Mathematics from Cantor through Russell to Gödel.* (Princeton, NJ: Princeton University Press, 2002); Paolo Mancosu, *From Brouwer to Hilbert: the Debate on the Foundations of Mathematics in the 1920s* (Oxford, UK: Oxford University Press, 1998); Loren Graham, Jean-Michel Kantor, *Naming Infinity: A True Story of Religious Mysticism and Mathematical Creativity* (Cambridge, MA: Belknap Press, 2009), esp. pp. 19 - 32; Morris Kline, "The Foundations of Mathematics" in *Modern Mathematics*, Vol. 3 (1972): 1182 - 1211; Apostolos Doxiadis and Christos Papadimitriou, *Logicomix: An Epic Search for Truth* (London, U.K.: Bloomsbury: 2009); David Rowe, "Anxiety and Abstraction in Nineteenth-Century Mathematics" in *Science in Context*, Vol. 17, No. 1/2 (2004): 23 - 47. Several accounts of the foundations crisis are given in tandem with historical biography of key participants. Including: Ray Monk, *Ludwig Wittgenstein: The Duty of Genius* (London, U.K.: Jonathan Cape: 1990); Monk, *Bertrand Russell: The Spirit of Solitude 1872 - 1921* (New York, NY: Free Press, 1996); Dirk van Dalen, *Mystic Geometer, and Intuitionist: The Life of L.E.J. Brouwer* (Clarendon Press, 1999); Constance Reid, *Hilbert* (Berlin, Germany: Springer-Verlag, 1970); Joseph Dauben, *Georg Cantor: His Mathematics and Philosophy of the Infinite* (Princeton, NJ: Princeton University Press, 1990). I am very grateful for conversations with Alma Steingart and Joan Richards about this material and the Foundations Crisis during out general exam in history of mathematics.

In order to salvage mathematics from these troubling contradictions and diffusions, certain communities of mathematicians and logicians set out in search of new foundations that could be used to build mathematics from the bottom up, eliminating the possibility of contradiction, and providing justification for mathematical truth claims. They wanted to put mathematics in one place and use the same grounding to justify and present all mathematical truths. Peano, and a community of other mathematicians working at the turn of the twentieth century, believed that *logic* was the answer.[38] It was a formal system in which they hoped all of mathematics could be reliably put together and justified.[39]

Two English mathematicians - Alfred North Whitehead and Bertrand Russell - became influential advocates for this view. Early in the twentieth century, they took Peano's project of theorem-collection one step further. They sought not only to collect,

[38]Mathematical logic has a rich and complex history - intimately bound up with the history of algebra - and it consists of many subfields, each carved out through the definition of the basic logical units in question and the operations that can be applied to them. Gregory Moore offers an account of how first-order logic (the propositional calculus with the addition of quantification of the from "for all" - $\forall$ - and "there exists" - $\exists$- in "The Emergence of First-Order Logic" in *History and Philosophy of Modern Mathematics*, Vol. 11 (1988): 95 - 135. Kline offers a succinct account of the origins of mathematical logic in the nineteenth century, drawing on developments from as far back as Renée Descartes and Gottfried Wilhelm Leibniz. (see Kline, "The Rise of Mathematical Logic" in *Mathematical Thought From Ancient to Modern Times*, Vol. 3 (New York: Oxford University Press, 1972): 1187 - 1192). Joan Richards offers an account of the cultural and ideological complexity of Augustus de Morgan's contribution to algebra and logic in "Augustus de Morgan, the History of Mathematics, and the Foundations of Algebra" in *Isis*, Vol. 78, No. 1 (March 1987): 6 - 30. Richards also explores some relations between the development of mathematical logic and the changing field of geometry in the nineteenth century in *Mathematical Visions: the Pursuit of Geometry in Victorian England* (Academic Press, 1988). See also Ernst Nagel, "The Formation of Modern Concepts of Formal Logic in the Development of Geometry" in *Osiris*, Vol. 7 (1939): 142 - 224; N. I. Styazhkin's traditional account of the history of logical "ideas and attitudes", *History of Mathematical Logic from Leibniz to Peano* (Cambridge, MA: MIT Press, 1969).

[39]A view now called "logicism." This view of mathematics was not shared by all mathematicians. Csiszar, for example, explores Henri Poincaré's objection to the project of "cataloguing" mathematical knowledge on utilitarian grounds. However, this imagination of mathematics was profoundly influential in the early twentieth century. Famously, David Hilbert advanced this program in his emphasis on the axiomatization of mathematics and his identification of central open problems in 1900. See Hilbert, "Mathematical Problems" in *Bulletin of the American Mathematical Society*, Vol. 8, No. 10 (1902): 437 - 479. Russell, Whitehead, and Peano were also engaged in a longer history of mathematics and logic that ran through Frege, Boole, and even Leibniz, that sought a universal system for representing and working in mathematics.

standardize, and circulate just theorems but *proofs* of those theorems as well. Their hope was to construct a "complete enumeration of all ideas and steps in reasoning employed in mathematics."[40] They wanted an explicit and comprehensive compendium of the "primitive ideas" and "primitive propositions" that ground mathematical reasoning, and to represent each established mathematical result as a "chain" of those basic elements.

The product of their efforts was the still-canonical text in mathematical logic, *Principia Mathematica.*[41] It's three volumes were first published by Cambridge University Press in 1910, 1912, and 1913. In them, Whitehead and Russell attempted (though never succeeded) to provide fully formalized deductive proofs of existing mathematical truths within their formal deductive system so that they could be known with logical certainty.[42] They outlined six "primitive ideas" and ten "primitive propositions" that

[40] Alfred North Whitehead, Bertrand Russell, *Principia Mathematica*, Vols. I - III. (Cambridge, UK: Cambridge University Press, 1910, 1912, 1913). I am here relying on Whitehead, Russel, *Principia Mathematica*, Vols. I - III (Rough Draft Printing, 2011 [1910 - 1913]). The Rough Draft editions are published digitizations - unedited and unabridged - of the first edition of the three volumes of *Principia* originally published in 1910, 1912, and 1913 respectively. The digitization work was done by Watchmaker Publishing. Whitehead, Russell, *Principia Mathematica*, Vol. 1: p. 3.

[41] Whitehead, Alfred North; Bertrand Russell. *Principia Mathematical*, Vols. I - III. Cambridge, U.K.: Cambridge University Press, 1910, 1912, 1913. In what follows, I worked with a reprinting of the first edition of Volume One digitized by Watchmaker Publishing and printed by by Rough Draft Printing, 2011. The pagination and typesetting of this volume is a replication of the original.

[42] *Principia* was never completed as it was originally envisioned. Whitehead and Russell intended to include a section on the logical foundations of geometry that was never written. See Russell, *My Philosophical Development* (London, U.K.: Allen and Unwin, 1959), on p. 99. The project laid out in *Principia* was perhaps most famously taken up by German mathematician David Hilbert. Hilbert also advocated for the translation of the branches of mathematics into formal, deductive, axiomatic systems. The project he laid out, often called "Hilbert's Program" became a central tenet of modern, twentieth century mathematics. See Richard Zach, "Hilbert's Program" in the *Stanford Encyclopedia of Philosophy* (July 2003: http://plato.stanford.edu/entries/hilbert-program/) for a discussion of the program. See also David Hilbert, Ackermann, *Principles of Mathematical Logic*, trans. L. Hammond, G. Keckie, R. Steinhardt (New York, NY: Chelsea Publishing Company, 1950 [1928]). The view that logic was the proper and complete foundation for mathematics was irreparably undercut in the 1930s by the likes of Turing and mathematician Kurt Gödel who studied crucial limitations of formal systems. They showed that the hope of reducing all of mathematics to logic was, in fact, impossible. Nonetheless, many mathematicians and logicians continued to ask questions about what can be deduced from what, and what kinds of problems could be solved within what kinds of formal systems. *Principia* and Hilbert's *Principles of Mathematical Logic* were and remain canonical texts grounding these studies. Kurt Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica

would serve as the definitions, axioms or premises of their logical system.[43] Among the primitive ideas were included basic concepts like "elementary proposition" and "assertions" as well as the definitions of the basic logical operators that would constitute their system, like "disjunction" and "negation". The primitive propositions were meant to be the clear and basic consequences of the primitive ideas including, for example, propositions like "Anything implied by a true elementary proposition is true" and "If q is true then "p *or* q" is true."[44]

From these primitives, Whitehead and Russell craft the basic rules for transforming and manipulating propositions: the rules of logical inference. They were designed to be immediately obvious, capturing the basic steps of deductive reasoning.[45] Included here, for example, is the so-called "principle of simplification" presented in their notation as "$\vdash : q \ . \ \supset . \ p \supset q$" which reads "$q$ implies that p implies q, *i.e.* a true proposition is implied by any proposition."[46] Also included here is the famed logical "syllogism" stated in their notation as "$\vdash : \ . \ q \supset r \ . \ \supset : \ p \supset q \ . \ \supset \ . \ p \supset r$" which reads "if r follows from q, then if $q$follows from p, $r$follows from p."[47] After laying out these primitives, Russell and Whitehead "proceed to formal deductions" which occupy the rest of their three volumes. These primitive ideas, propositions, and rules of inference constitute the formal infrastructure of *Principia* that constituted a particular approach

und verwandter System I." *Monatshefte für Mathematik* Vol. 38, No. 1 (1931): 173 - 198; Turing, "On Computable Numbers with an Application to the Entscheidungsprobelem" in *Proceedings of the London Mathematical Society* Vol. 42 (1936): 230 - 265; Alonzo Church, for example "An Unsolvable Problem in Number Theory" in *American Journal of Mathematics* Vol. 58, No. 2 (1936): 345 - 363. I discuss their work in more depth in Chapter Two "Mathematical Objects in Action."

[43]These are detailed in Section A "The Theory of Deduction," Part 1 "Primitive Ideas and Propositions," pp. 94 - 101. In the second edition of Volume One, the ten "primitive propositions" were distilled to seven, and presented with markedly less exposition.

[44]Whitehead, Russell, p. 98 - 99.

[45]Presented in Whitehead, Russell, *Principia Mathematica*, Volume One, Part A, Section 2 "Immediate Consequences of the Primitive Propositions," pp. 101 - 113. These rules are also distilled in the second 1963 edition of *Principia*, published on pp. 13 - 14.

[46]Whitehead, Russell, *Principia Mathematical*, Vol. 1, p. 103.

[47]Whitehead, Russell, *Principia Mathematical*, Vol. 1, p. 103.

to theorem-proving.[48]

Whitehead, and Russell's project reflected a particular philosophy of mathematical knowledge - that it could be fully formalized, axiomatized, and standardized within mathematical logic. For them, proofs were chains of deductive steps that began with the axioms of their logical system and concluded with a true and interesting logical statement, and they believed the whole of mathematical knowledge could be formulated in this way.

As will be discussed in the section "A Mathematical World on Paper" in Chapter One, *Principia* was also constituted by a particular set of material tools and practices. The formal system just described was also accompanied by a written symbol system meant to make those primitive ideas easy to see, easy to read, easy to follow line by line on the page. Logical propositions and inference rules were tethered to a hand-written and typeset symbol system, designed to enable the heads and hands of mathematicians to pursue the work of proof-as-logical-deduction envisioned by people like Whitehead and Russell. These pages of proofs would be compiled in *books*. *Principia*, like Peano's Encyclopedia of proofs that preceded it, were mean to collect mathematical knowledge, written down, standardized both formally and representationally and circulated to the growing community of professional mathematicians. It was a paper-based culture; a book culture.

Automated theorem-proving was one of the inheritors of this early twentieth century tradition of mathematics. This culture of proof came with a set of particular formal and material tools. And these, I claim, were precisely what is at stake in the

[48]These basic elements of logic would be distilled and given a more "modern" formulation by David Hilbert and Wilhelm Ackermann in *Grundzüge Der Theoretischen Logik*. Berlin, Germany: Springer Verlan, 1928 [*Principles of Mathematical Logic*, trans. L. Hammond, G. Keckie, R. Steinhardt. New York, NY: Chelsea Publishing Company, 1950]. Many of those working on the automation of proof in the 1950s and 60s made use of the the later Hilbert-Ackermann formulation rather than working directly from *Principia*.

proof automation attempts examined in this dissertation. New rules of inference, new representation systems, new formalisms, new materials would be introduced to the work of logical proof, intervening in the character of deduction and logic itself, though perhaps not in the way one would expect.

Computers could only do what they could be explicitly instructed to do. As such, the project outlined in *Principia* seemed a perfect place to introduce computers to the work of proof - *Principia* came with a ready made set of axioms and explicit rules of inference. As such, an intuitive way to approach the automation of proof would be to provide those axioms and permissible inference rules to the computer. The computer could then be programmed to apply the inference rules to the axioms in order to deduce any provable logical consequences. Users could then input some mathematical proposition, P, and run the computer to see if any permitted sequences of inference led to P. If so, the series of steps taken by the program in that sequence would constitute a proof of P.

However, in spite of the incredible speed and efficiency with which computers (even in the second half of the 20th century) could execute instructions, automated theorem-proving practitioners quickly discovered that this method of proof-seeking on its own was so inefficient as to be unusable. Not only did it lead to an *exponential* explosion of data given how many inferences could be made, but there was also no way to know when or if a proof would ever be found. This exhaustive method guaranteed that every provable statement P would eventually be proven, but it was not clear that this would happen during the lifetime of mathematicians who actually cared about the problem.

Larry Wos, a member of the automated theorem-proving team at Argonne, wrote the following of this approach in 1964: "All permissible substitutions were made in a systemic, exhaustive manner, guaranteeing that if a proof existed of the desired theorem, it would be captured in the steadily expanding sets of instances. The disastrous

rate of growth of these sets, due to the inclusion of numerous unprofitable inferences, spelled the doom of exhaustive instantiation".[49] Merely instructing computers to apply inference rules to axioms was not a feasible way to harness the power of computing for theorem-proving. *Principia* could not be automated as-is.

As such, automated theorem-proving practitioners instead sought to develop mechanisms that would restrict exhaustive search and cut down on the number of 'unprofitable inferences'. But the mechanisms they designed took quite different forms: some were inspired by human practice and others were based in the structure of logical systems themselves. The three programs I discuss in the dissertation each followed a different set of protocols to avoid an intractable explosion of data.

Newell, Shaw, and Simon designed the Logic Theory Machine in the image of a human mathematician. They believed that people rely on their intuitions to choose particular paths of deductive and avoid the overwhelming explosion of data that comes from deducing all possible conclusions. They wanted to identify, formalize, and ultimately automate the kinds of intuitions or, *heuristics*, they believed people used in selecting paths of inference.

Hao Wang was uninterested in human tricks because he believed computers and humans to be capable of quite different ways of searching for a proof. Instead he wanted to know what possible tricks might come out of the study of logic itself. In particular, he developed a method by which computers could look for patterns that could only be recognized at scales that were inaccessible to human perception. It would use those patterns to cut through the explosion of consequences in search of proofs.

The team at the Argonne National laboratory agreed and disagreed with both of these approaches. They wanted to make use of human intuitions, but they believed

[49]Larry Wos. "The Unit Preference Strategy in Theorem Proving", in *The Collected Works of Larry Wos*, Vol. 1 (Hackensack, NJ: World Scientific Publishing Company, 2001), pp. 17 - 28 [originally *American Federation of Information Processing Society*, Proceedings 26 (1964), pp. 615 - 621].

computers could never be made to possess them. They also believed that computers could wield more powerful rules of inference than their human counterparts. So they designed collaborative theorem-proving programs in which human intuitions would point the computer down certain deductive paths but the computer would follow those paths by executing inferential steps that would be difficult for people to take.

These three approaches also served as exemplars of the way that later research would address this problem. They were formalized by automated theorem-proving researchers in the early 1980s as categories for organizing early work in the field. In reflecting on the first quarter-century of automated theorem-proving work, David Loveland identified two approaches - the "Logic" approach and the "Human Simulation" simulation approach, with collaborative programs situated in between them.[50] The logical approach looked for new logically sound ways of looking for deductive proofs that might be impossible for people to use because of their complexity. The human simulation approach looked instead to identify and automate what people do. And the collaborative approach sought to combine complex logical processing with un-automated human intuiting.

These approaches were also intended to produce different kinds of *proof*. The human simulation approach was intended to produce proofs *of the kind people might make*, following paths that people might follow and taking steps of the kind that people take. The logic approach, on the other hand, produced proofs that might be difficult for people to understand because they are constituted by steps that people may not be able to take and because they might be so long people would not be able to read them. The former proofs were to be trusted because people could read them and see how they work. The latter proofs were to be trusted if the computer could be trusted to

[50]Donald Loveland, "Automated Theorem-Proving: A Quarter-Century Review" in *Automated Theorem-Proving: After 25 Years* [*Contemporary Mathematics*, Vol. 29] (Providence, RI: The American Mathematical Society, 1983): 1 -46.

accurately execute rules known to be *sound*, even if they were not surveyable to a human reader. Both kinds of proof diverged from the step by step deductive chains presented in *Principia* and thought to epitomize the soundness of inferential reasoning. But some people *didn't* trust proofs of one or the other (or either) kind.

Historian of science and technology Donald MacKenzie has argued that automated theorem-proving animated a debate about what proofs should be like.[51] In particular, he argues that new cultures of proof emerged along two axes. The first axis concerned the *kind* of proof being produced, which could be either "formal" or "rigorous," the latter also known as "informal." MacKenzie, drawing from the earlier work of Eric Livingston, differentiates between these two kinds of proof as follows:

> A formal proof is a finite sequence of 'well-formed' (that is, to put it loosely, syntactically correct) formulae leading to the theorem, in which each formula is an axiom of the formal system being used or is derived from previous formulae by application of the system's rules of logical inference. [...] Rigorous arguments, in contrast, are those arguments that are accepted by mathematicians (or other relevant specialists) as constituting mathematical

[51]See MacKenzie, "Computing and the Cultures of Proving" in *Philosophical Transactions of the Royal Society A: Mathematical, Physical, and Engineering Sciences*, Vol. 363 (2005): 2335 - 2350. MacKenzie also explored several early automated theorem-proving programs and the debates that emerged surrounding their design and their results. Much of MacKenzie's work, especially in *Mechanizing Proof: Computing, Risk and Trust*, is in fact about a particularly important application of automated theorem-proving software to a field called "program correctness proof" in which the goal was to prove theorems *about programs.* Computers came to occupy increasingly important positions in military, aerospace, banking, and political infrastructure in the United States, it became increasingly important that computer programs behave as they were intended. Trial and error testing - the dominant method in early computing - was limited by the programmers' ability to accurately and comprehensively foresee possible failings. This method was deemed unreliable, and many communities around the United States aimed instead to represent computer programs as mathematical structures and then to *prove* that they behaved in the desired way, e.g. would always terminate, or could never reach an undesirable state, etc. Computer programs would be as reliable as the mathematical knowledge that could be forged about them. However, MacKenzie suggests that in fact mathematical knowledge is not the bedrock of self-evident certainty that it has often been held up to be, but is rather the product of complex social negotiations of credibility, trust, and risk. See MacKenzie, *Mechanizing Proof; Computing, Risk, and Trust.* MacKenzie also explores how the emergence of computer proof led to the establishment of novel trust relations between mathematicians, programmers, government officials, the public, and military personnel as computer proof was mobilized as a guarantor of proper software function.

> proofs, but that are not formal proofs in the above sense.[52]

Formal proofs are often characterized as showing *that* a theorem is true - showing that it follows from a given set of axioms. Rigorous proofs are often characterized instead as showing *why* something is true - foregrounding some crucial insight that convinces readers that the theorem is true even in the absence of an exhaustive deduction. These two kinds of proof existed before computers, but MacKenzie argues that mathematical communities polarized around them in new ways when computers came on theorem-proving scene. Certain automated theorem-proving practitioners tried to harness computing to produce one kind of proof or the other. And mathematicians, philosophers, and other technical practitioners would accept and reject computer proofs based on allegiance to one or the other kind of proof. Proof, as MacKenzie has compelling shown time and again, is inherently *social* in character: different communities decide what does and does not convince them.[53]

This dissertation builds on MacKenzie's account to show that the constituent elements of those two kinds of proof - form and rigor, inference rules, intuitions and crucial insights - came to *mean different things* with the introduction of computers. In implementing their software, theorem-proving practitioners gave new meaning and new manifestations to some of the existing central tenets around which debates about proof emerged in their wake.

New inference rules were devised. Intuitions were both represented in new ways and

[52]MacKenzie, "Computing and the Cultures of Proving," p. 2338.

[53]Other important accounts of the social character of mathematics include David Bloor, "What Can the Sociologist of Knowledge Say About 2 + 2 = 4" in *Mathematics, Education, and Philosophy: An International Perspective*, ed. Paul Ernest (London, UK: The Falmer Press, 1994): 21 - 32; Bloor, "The Living Foundations of Mathematics" in *Social Studies of Science*, Vol. 17, No. 2 (1987): 337-358; and Henk Bos, Herbert Mehrtens, "The Interactions of Mathematics and Society in History and Some Exploratory Remarks" in *Historia Mathematica*, Vol. 4, No. 1 (1977): 7 - 30; Charles Fisher, "The Death of a Mathematical Theory: A Study in the Sociology of Knowledge" in *Archives for History of Exact Sciences*, Vol. 3 (1966): 137 - 159; Sal Restivo, *Mathematics in Society and History: Sociological Inquiries* (Dortrecht, Netherlands: Kluwer, 1992).

generated by new experiences. In practice, the automation of proof changed the formal and material character of the concepts around which cultures of proof materialized. By focusing on implementation I show how the introduction of computing to proof not only prompted the emergence of communities newly polarized on the axis of formal vs. informal proofs. It also changed the *meaning* and the *constitution* of concepts like form, rigor, intuition, and inference that informed those two cultures of proof.[54]

For example, each of the three programs included a different formalization of the four canonical rules of inference presented in Whitehead and Russell's *Principia*.[55] In the Logic Theory Machine, the four canonical rules of inference from *Principia* were translated into forty-two variations of eight computer operations for manipulating what were called "linked list data structures," which were in turn a new representational system for the logical propositions contained in *Principia*. Hao Wang, in designing

[54]More generally, MacKenzie and sociologist Claude Rosental have explored the advent of so-called "computer proof" as a window into the irreducibly *social* character of mathematical knowledge and both demonstrate how computers changed the stakes of existing social practices of mathematical knowledge-making. See Claude Rosental, *Weaving Self-Evidence; A Sociology of Logic* (Princeton, NJ: Princeton University Press, 2008); Rosental, "Certifying Knowledge: The Sociology of a Logical Theorem in Artificial Intelligence" in *American Sociological Review*, Vol. 68 (2003): 623 - 644; Donald MacKenzie, *Mechanizing Proof: Computing Risk and Trust* (Cambridge, MA: The MIT Press, 2004); MacKenzie, "The automation of proof: A historical and sociological exploration" in *IEEE Annals of the History of Computing*, Vol. 17, No. 3 (1995): 7 - 29; MacKenzie, "Slaying the Kraken: The sociohistory of a mathematical proof" in *Social Studies of Science* Vol. 29, No. 1 (1999): 7 - 60. Rosental tracks the production, debate about, and ultimate accreditation of a computer proof of a logical theorem, attributed to the programmer Charles Elkan in the 1990s. He argues ultimately that what is thought to be "self-evident" in logic - immediately accessible to any mind capable of "right reasoning" - is in fact the product of complex social negotiations. The theorem "was not evaluated collectively through the aggregation of attentive readings of the proof presented in [the] article" nor by the "systematic application of universal criteria for evaluating a proof" (Rosental, "Certifying Knowledge," p. 639, 640). Instead, the participants were "social actors, part of long-term and large-scale conflicts, sometimes engaged in rival activities and projects, endowed with variable resources and competencies." (p. 640). Crucially, the debate about Elkan's proof was *also* a debate about how best to approach the field of Artificial Intelligence (AI). Elkan was an advocate for what was called "Classical AI" and many of his critics and doubters of his result came from emerging "rival theories" like "fuzzy logic," "neural networks," "genetic algorithms," and so on (p. 628). Rosental demonstrates how competing approaches to AI, competition for AI funding resources, and underlying disagreements about the character of the computer and the prospects of computer thinking were incorporated into existing processes of accreditation in mathematics.

[55]See also Hilbert, Ackermann, *Principles of Mathematical Logic*. Most automated theorem-proving researchers used Hilbert's formulation rather than the original.

the System P (predecessor to the Program P), translated those rules of inference into eleven computer operations for iteratively removing logical operators. And John Alan Robinson invented a new rule of inference called the Resolution Principle which, all on its own, could replace all four of *Principia*'s inference rules. That rule was build in to the computer side of the collaborative AURA program to be discussed in Chapter Three. To be sure, at a high enough level of abstraction these sets of rules are all equivalent. They can be shown

The question that motivates me here is, in a sense, *prior* to the one MacKenzie asked. For him, what is interesting is how inclusions and exclusions were performed by different communities in response to so-called computer proofs - how new priorities and cultures developed around computers as potential contributors to mathematical knowledge. I propose that something was already happening to proof before there were any so-called computer proofs on the table to negotiate and disagree about.

I want to know what was actually involved in introducing computers to the work of proof in the first place - when early theorem-proving software was being initially designed and implemented. Relevant mathematical objects had to be turned into things that could be input to computers and stored and manipulated in its electromagnetic storage systems. Processes of theorem-proving had to be translated into algorithmic, rule bound, electronic computational operations. And that meant taking what was written in the pages of *Principia* and the *Principles of Mathematical Logic* and, to borrow again from Mahoney, "putting them into the computer."

The new forms of proof that emerged were constituted by *practices*, including obviously programming, but also new ways of writing, diagramming, thinking, and doing that went into the design and implementation of theorem-proving programs. These new forms of proof were also constituted by new *materials* - those that constituted computing technology. And these new forms of proof were constituted by new *for-*

malisms - those algorithms, programming languages, and representation systems that translated the work of proof into computable processes.

Each of the three theorem-proving programs I explore here took up the early twentieth-century tradition of mathematical logic. They work within logical systems, they engage in logical inference, they even prove the same theorems as Whitehead and Russell prove in their canonical logical texts. They were designed to do the work of logical proof. However, each program also intervenes in that tradition and that is my focus in the discussions to follow. The elements of proof - intuition, inference principles, form, symbolism - were given new meaning in the work of automated theorem-proving. And they were given different new meanings in the hands of practitioners who wanted to use computers in different ways. Through their efforts, proof came in new forms.

The Possibilities of Computing

In spite of their many differences of opinion, automated theorem-proving practitioners also shared certain key perspectives. None of them wanted to use computers as mere brute force "deducers" but rather wanted to use them to find more interesting or more effective ways to search for proofs. This belief was part of a more general perspective: like many early computing practitioners, automated theorem-proving researchers believed the computer was a more interesting kind of tool than it was at first imagined.

The first reprogrammable digital computers were built to *calculate*: they were both functionally intended and conceptually understood as numerical data processors.[56] Computers were used for the calculation of complex ballistics trajectories, for solv-

[56]This was especially true in the United States. In Britain, practitioners like Alan Turing imagined and actualized nonnumeric uses for computers very early on. Even the language decryption work Turing and his colleagues engaged in at Bletchley Park during the Second World War could be seen as nonnumeric in character. This was less the case in the United States where the early computers were designed and used with numerical calculation in mind. I am grateful to conversations with historian of computing Marie Hicks for this difference in national computing styles.

ing differential equations, for the numerical simulations of nuclear chain reactions, and so on.[57] Very much in keeping with their closest technological ancestors - tabulating machines, punched card readers, and other electric and mechanical calculators - this was the case overwhelmingly for early computing machines. Numbers were input to the computer, instructions were given for the functional and arithmetic manipulation and storage of numbers, and numeric values were the output as well.

Recall that Turing wanted a more "intelligent" kind of behavior from his computers than mere calculation. So too did practitioners of automated theorem-proving. They were among those computing research pioneers from certain other fields who developed a different understanding of what computers could do, of what kind of thing they were. These practitioners pursued a vision of computers as *symbol processing machines*, capable of manipulating any formal system whatever - whether it referred to numbers or not.[58]

[57]In fact, the earliest mainframe computers were designed to replace human computers who performed complex numerical calculations through systems of labor division. See David Alan Grier, *When Computers Were Human* (Princeton: Princeton, 2005) for a discussion of the original human meaning of the word "computer." See also Harry Polachek, "Before the ENIAC" in *IEEE Annals of the History of Computing* Vol. 19 No. 2 (1997): 25 - 30. His account focuses on the human calculators that preceded the ENIAC and unpacks the complexity of the calculations and the labor distribution in response to which the ENIAC was designed. Examples of practitioner reports that detail early mainframes explicitly as numerical entities designed to accept numerical data as input and to perform numerical calculations include Herman Goldstine, Adele Goldstine. "The Electronic Numerical Integrator and Computer (ENIAC)" in *Mathematical Tables and Other Aids to Computation*, Vol. 2, No. 15 (1946) 97 - 110: 98; Douglas Hartree, "The ENIAC, An Electronic Computing Machine" in *Nature* Vol. 158 (1946): 500 - 506; John von Neumann, "First Draft of a Report on the EDVAC" Contract No. W-670-ORD-4962 between the United States Army Ordnance Department and the University of Pennsylvania. Document dated June 30, 1945. Document has been transcribed and edited by Michael D. Godfrey, November 1992.

[58]It bears noting that in one sense, digital electronic computers are unavoidably numerical at base. Information is always stored in memory in numeric form, e.g. hexadecimal notation, which is specified by bits capable of having one of two possible states. Computers operate according to binary operations based on the presence and absence of electrical current, or the orientation of magnetic fields, understood as corresponding to the 1s and 0s, or True's and False's of Boolean logic. In both of these senses, the mechanism for computing is numerical. This transition from numeric to nonnumeric processing does not claim otherwise. Rather, this transition opens up possibilities that what is *referred to* by those numbers in memory or by those bits of information is not numeric. Systems of correspondence are worked out and interfaces are devised that enable users to input nonnumeric information (even typing on a keyboard is an example of this) and to receive nonnumeric information out of the

This was an ontological issue. It was about what kind of thing the computer *is* and what kind of work it could do. Automated theorem-proving practitioners and others like them turned computers into a new *kind* of thing by thinking about them and using them as machines that could do nonnumerical mathematical work. In their retooling of the computer, in their translation of objects and processes of mathematics into computable form, in their conceptualizing and crafting of the computer as a tool, an object, an agent - these practitioners are all seeking to articulate and implement a new vision of the computer-as-object.[59]

This perspective was also related to the emergence of computer science as an academic discipline. Nonnumeric and symbolic computing was historically bound up with the development of programmatic and theoretical studies of algorithms and data-structures.[60] In that capacity, it was also entwined with the emergence of practitioner claims that the computer was a thing worthy of study in its own right, rather than a mere handmaiden or tool for other discipline's data processing needs.

Automated theorem-proving was therefore not merely an esoteric niche of automation research but, it rather participated in the development of perspectives, tools, and vocabularies that were enlisted in the program of elevating computer science to the status of a discipline. In fact, many practitioners that worked on the automation of mathematics were very active participants in the establishment of computer science

computer (all graphical user interface - including the appearance of words on the screen - are examples of this). For example, computers could be "word processors". Users input alphabetical information to the computer through a keyboard and the computer can display and manipulate it through a graphical user interface. Even though "under the hood" the information was represented by numerical data, the computer was performing more than numerical processing - it performed "word processing". The crucial insight is that in nonnumeric processing, the instructions given to the computer do not correspond to well-defined mathematical functions for numbers.

[59]For a synthetic account of the emergence of symbolic computation, see Edward Ng, "Symbolic-Numeric Interface: A Review" NASA Jet-Propulsion Lab, Technical Report N80-16768 (January 1, 1980).

[60]For example, see Peter Brillinger, Doran Cohen, *Introduction to data structures and non-numeric computation* (Prentice-Hall Series in Automatic Computation, 1972); Patrick Hall, *Computational Structures: An Introduction to non-numeric computing* (MacDonald and Janes, 1975).

departments.

Moreover, all practitioners of automated theorem-proving believed that computers could *surprise* us. Some early skepticism about the possibilities of computing were grounded on the observation that computers can only do just exactly what they are instructed to do, and that they would therefore never perform beyond what their programmers could imagine for them. However, most early computing practitioners insisted that it was often not possible to know what the consequences of their instructions would be. Indeed, if they could, they would have had little need for fast computing machinery to carry them out. Herbert Simon wrote in 1960 that "This statement - that computers can only do what they are programmed to do - is intuitively obvious, indubitably true, and supports none of the implications that are commonly drawn from it."[61]

In response to those erroneous implications, two early Artificial Intelligence practitioners Edward Feigenbaum and Julian Feldman summarized the position of most computing practitioners like this:

> [I]t is wrong to conclude that a computer can exhibit behavior no more intelligent than its human programmer and that this astute gentleman can accurately predict the behavior of his program. These conclusions ignore the enormous complexity of information processing possible in problem-solving and learning machines. They presume that, because the programmer can write down (as programs) general prescriptions for adaptive behavior in such mechanisms, he can comprehend the remote consequences of these mechanisms after the execution of millions of information processing operations.[62]

These sentiments were shared by the practitioners I discuss in this dissertation and in

[61]Herbert Simon, *The New Sciences of Management Decision* (New York, NY: Harper & Row, 1960), p. 25.

[62]Edward Feignebaum, Julien Feldman, "Artificial Intelligence" in *Computers and Thought* (New York, NY: McGraw Hill, 1963), p. 4. This book is a volume of early contributions to Artificial Intelligence compiled and edited by Feigenbaum and Feldman to serve as a kind of early text book of fundamentals. It included a section on "machines that prove theorems," in which were featured the Logic Theory Machine and its closest descendent, the Geometry Theorem Proving Machine.

early computing fields more generally.

Computers, it seemed, were something of a paradox. On one hand they were the most "disciplined" of technologies - they were rule bound to a fault, executing human-crafted instructions to a *T*. And at the same time, their ability to execute human instructions exceeded the capacity of the human instructors to anticipate. This is both why *experimentation* and *empiricism* came hand in hand with the use of computers - often it is not until actually running a program that one discovers what consequences lurk in a set of instructions. Moreover, it is another reason why implementation is significant - because abstract algorithms and computational models don't run on computers they do not offer up any of those unforeseeable consequences.

In this regard, computers resemble "experimental systems" as formulated by Hans-Jörg Rheinberger - the material and physical systems that enable scientific research.[63] That concept captures the way in which even the most disciplined of scientific instruments or material apparatuses possess "inherent unpredictability" - they go beyond formal prescriptions, they can surprise their users, "they contain the possibility of an *excess*. They contain more and other possibilities than those to which they are actually held to be bound."[64] For Rheinberger, the production of novel scientific knowledge is incumbent upon that unpredictability belonging to "material objects" the latter being "driving forces in the process of knowledge acquisition."[65]

[63]Hans-Jörg Rheinberger, "Experimental Systems: Historiality, Narration, and Deconstruction" in *Science in Context*, Vol. 7, No. 1 (1994): 65 - 81; Rheinberger, *Toward a History of Epistemic Things: Synthesizing Proteins in the Test Tube* (Stanford, CA: Stanford University Press, 1997).

[64]Here he notes the seeming tension between formalization and unpredictability for scientific knowledge-production in general, proposing that "Although unpredictability is in the nature of scientific undertakings, their movement and performance can be characterized in a formal - i.e., structural - way," Rheinberger, "Experimental Systems," p. 70, p. 71, emphasis in original.

[65]Rheinberger, "A Reply to David Bloor: 'Toward a Sociology of Epistemic Things"' in *Perspectives on Science*, Vol. 13, No. 3 (2005): 406 - 410, p. 406. Rheinberger's claim should be understood as part of a broader discussion concerning the role of nonhuman agency in scientific knowledge production. He allies his project with actor network theory, proposed most famously by Bruno Latour, which resists accounts that seek to reduce scientific knowledge-production to social (read, human) factors alone.

However, Rheinberger seeks to differentiate experimental systems, and the *epistemic things* that are manipulated and produced within them, from *technical things*. These latter are "transparent, confined," stable, not unpredictable, or as Rheinberger puts it "not transcendent."[66] Once they have run out of productive indeterminism, epistemic things can be made in to technical things, secondary and stable, and these in turn can be built into new experimental systems. Here, I diverge from Rheinberger. I argue that technical things can be epistemic things, and that the project of producing technical things, like computer programs, can produce novelty and surprise like work in an experimental system. Indeed, this is another way of saying that implementation has epistemological significance. I identify in computers and computer programs the kind of indeterminacy that Rheinberger reserves for experimental systems, and in so doing, I seek to bridge the technical and the scientific, and with it, the history of technology and the history of science.

Here, I similarly diverge from Bruno Latour, Rheinberger's colleague in advocating for nonhuman agency. Within actor network theory, Latour sought to distinguish between epistemologically significant *mediators* that "transform, translate, distort, and modify the meaning or the elements they are supposed to carry" and *intermediaries* that transport "meaning or force without transformation: defining its inputs is enough to define its outputs."[67] Intermediaries are stable, completely defined. They do not overflow, they do not surprise, they are not epistemologically significant. Latour cites a "properly functioning computer" as a "good case of a complicated intermediary" - suggesting that computers need to break down in order for them to become transformative mediators. I disagree. All three of the properly functioning computers in this dissertation, and the fully formalized, completely defined computer programs run on

[66]Rheinberger, "A Reply to David Bloor," p. 407.

[67]Bruno Latour, *Reassembling the Social: An Introduction to Actor-Network Theory*, (Oxford, U.K.: Oxford University Press, 2005), p. 39.

them produced unexpected results, overflowed, contained an *excess*.

As we will see, each of the theorem-proving programs discussed in this dissertation surprised their developers, but each surprise was different. The Logic Theory Machine produced a previously unknown proof of a proposition from *Principia Mathematica.* Newell and Simon (and Russell) believed that it was more "elegant" than that Russell and Whitehead provided. They interpreted this as evidence that computers were capable of producing elegant new proofs of this kind logicians constructed. The Program P revealed what Wang called the "rather surprising result" that every proposition from *Principia Mathematica* in fact fell into a particular subset of the predicate calculus, a previously unknown result in mathematical logic. The Argonne team proposed that working with AURA in fact enabled them, much to their surprise, to collaborate in solving problems from mathematical domains about which they had only limited knowledge. They knew they wanted to work with the program to solve open problems but they didn't expect to be successful in problem-domains with which they were not familiar.

These unexpected results had different characters that reflected the different approaches to automated theorem-proving adopted by each designer. They also reflect differences in *what the designers were looking for.* That computers surprise their users still does not mean they *force* any particular conclusion or interpretation. The new results had to be identified, justified, and promoted by the users and developers. It does mean however, that part of the epistemological significance of computing emerges only once programs have actually been run and results inspected by programmers. This is another reason why implementing and running programs, not just modeling them, matters for the history of computing. Reformalisms produced new and unexpected results.

The cases discussed here, while specific and localized, speak to the *kinds* of transfor-

mations and the epistemic significance that accompanies the overwhelming proliferation of computing throughout the scientific and academic landscape, where today, it can be found everywhere. Obstacles that shaped the development of these programs - like developing tools for the management of very limited computer memory - were faced by all early computing practitioners. This dissertation thus offers a window into the character of early computing and the kinds of efforts expended there.

Each chapter that follows discusses one computer program, the approach to automated theorem-proving that motivated it, the disciplinary and political context that informed it, and the reformalisms that were enacted in its construction. The central focus is on the design and the implementation of each program and what I claim to the be epistemological novelties and associated forms of proof that emerged in tandem. I explore new ways of doing proof, and new tools for doing it with, that were developed as part of the implementation of those programs. I explore how different ideas about mind, about proof, and about computing were built into the infrastructure of each program and at how those ideas were in turn related to broader political, institutional, and disciplinary contexts. At the heart of each discussion is the question - what was known about mathematics and how was it known when practitioners thought and worked with the formal and material tools of modern digital computation.

The Logic Theory Machine, the Program P, and the AURA are sites in which proof was being translated into computing. This is a story about implementation. It is a story about how punched cards, algorithms, computer memory, user interfaces, programming languages, and data structures - the formal and material tools of computing - were mobilized for the purpose of representing mathematical objects, manipulating and exploring them, and ultimately for producing arguments about them whose status would ultimately be negotiated by mathematical communities. I track how mathematics and computers were simultaneously retooled in order to make the latter useful and

usable for the former - how new forms of mathematical proof and practice emerged in these implementation efforts.

Chapter 1

Rewriting *Principia*:

Implementing Intelligence

Introduction: Mathematical Materialities

As I write, there are upwards of 264 *exabytes* of digital storage, communication, and computation capacity available on the planet - in hard drives, servers, memory cards, flash drives, and other new media.[68] That is a number with twenty zeroes and one that exceeds the estimated number of grains of sand on the earth by more than 300 percent. This digital capacity exceeds our ability to store and circulate information using other media, like paper, by orders of magnitude. This is one reason why we are so often said to live in a digital age. A great deal of our world is housed in digital media.

These bytes and their constituent bits are used to encode, store, and manipulate all kinds of things from avatar gaming characters, photographs, and texts to scientific

[68]This number was put forward in 2011, and describes estimated digital storage capacity - including hard drives, memory cards, DVDs and so on, up to 2007. Martin Hilbert, Priscila Lopez, "The World's Technological Capacity to Store, Communicate, and Compute Information" in *Science* Vol. 332, No. 60 (April, 2011): 60 - 65, p. 60. This number was calculated using the modern standard in which each byte consists of eight bits.

models and massive databases. But on their own, bits are not meaningful. Bits don't tell us anything or do any work for us unless we know *how* they are being used to encode photos, texts, models, or anything else. In order to make computers useful for a given task, communities of users, programmers, and software designers must first develop ways of reformulating that task and its constituent elements so that they can be stored, accessed, and controlled in and by the computer. We only live in a digital age insofar as people have devised ways of storing and accessing the things they care about using digital media. As Michael Mahoney put it, the history of computing is in part a history of how different communities have "put their portion of the world into the computer."[69] And there is never any single way that communities must digitize their world. Instead, each community crafts or selects a particular set of encoding tools with which to translate pieces of their world into computational objects.

This chapter explores how one community put a small piece of the world of mathematics "into the computer." In particular, I investigate the development of an automated theorem-proving program called the Logic Theory Machine that ran on the Johnniac mainframe at the RAND Corporation in Santa Monica, California in the mid-1950s. The Logic Theory Machine was designed to produce proofs of theorems taken from the pages of an early twentieth-century canonical text in elementary mathematical logic - *Principia Mathematica*. In order to make this automation project possible, researchers at RAND transformed the elements of logic and the processes of theorem-proving into computational artifacts and operations.

I ask - what kind of work was involved in this transformation? What motivated this automation attempt? What kinds of obstacles and possibilities informed the process? How did these practitioners understand proof and think about mathematics in the

[69]Michael Mahoney, "What makes the history of software hard" in *IEEE Annals of the History of Computing* Vol. 30, No. 3 (2008): 8 - 18, p. 8.

context of computation? Ultimately, this chapter aims to recover and reconstruct new ways of representing the elements of logic that were devised in order to transport the world of *Principia* into the digital media of the Johnniac mainframe.

At bottom, this chapter is about the materials with which the work of mathematics is done. Traditionally, mathematics has been characterized as dealing with highly abstract and immaterial things. In fact, throughout history myriad material tools have been developed to make mathematics possible. From systems of written symbolic notation and diagramming to physical models and mechanical calculators, different technologies have equipped the heads and hands of mathematicians to formulate and explore their domain in different ways. Indeed, the world of *Principia Mathematica* that was "put into the Johnniac" in the context of the Logic Theory Machine was not simply an abstract immaterial world of logic that lay in wait of representation. It was a world on *paper* - a book world in which proofs were constructed on the page, typeset, bound, and circulated to communities of reading mathematicians.[70]

This chapter tracks a transformation from a human-oriented representational sys-

[70]In imagining the paper world of *Principia* I have in mind the rich and exciting scholarship that explores media, print, communication, and publication in history of science. I am particularly indebted to conversations with Alex Csiszar from whom I have learned a great deal about cultures of print, publication and the classification of mathematical knowledge at the turn of the twentieth century. See Adrian Johns, *The Nature of the Book: Print and Knowledge in the Making* (University of Chicago Press: 2000); Jim Secord, *Victorian Sensation: The Extraordinary Publication, Reception, and Secret Authorship of Vestiges of the Natural History of Creation* (University of Chicago Press: 2003); Lisa Gitelman, *Scripts, Groves, and Writing Machines: Representing Technology in the Edison Era* (Stanford University Press, 2000). Some scholars are interesting in looking at digital storage and recording mechanisms as episodes in history of *writing* - taking seriously the metaphor that information is "written" to discs and drives. I follow them in exploring digital storage as an episode in mathematical inscription and representation. In this regard, I follow media theorist Matthew Kirschenbaum, *Mechanisms: New Media and the Forensic Imagination* (The MIT Press, 2012) - he treats digital storage as a form of literal writing. More I am eager to explore new forms of paper and pencil writing and diagramming that were developed in the process of designing and implementing computer programs. I was excited to learn of similar work from like-minded historians of computing Mark Preistly, David Norfre, and Gerard Alberts in "When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950 - 1960" in *Technology and Culture,* Vol. 55, No. 1 (January 2014): 40 - 75, who explore different types of writing and representation in early programming practice. I explore some of these ideas in Dick, "Machines Who Write," in *IEEE Annals of the History of Computing*, Vol. 35, No. 2 (April-June 2013): 85 - 87.

tem to a machine-oriented one, keeping an eye on where and how materiality matters for questions of mathematical agency and knowledge-production. In the context of *Principia Mathematica*, logical propositions were inscribed on the page using a symbol system intended explicitly to capitalize on the human powers of vision and pattern recognition. I begin the chapter with an exploration of this paper world. The rest of the chapter tracks how new ways of representing logical propositions were developed - ones that accommodated a computer rather than a human practitioner.

In the context of the Logic Theory Machine, logical propositions were transported into the magnetic drum and core magnetic storage systems of the Johnniac mainframe in the form of what were called "Linked List Information Structures." These were an early example of what we now call "data structures" - one of the tools computing practitioners have devised to "put their portion of the world into the computer."[71] They are ways of organizing information in computer memory, of encoding things as digital things, of assigning meaning to underlying bits, of making digital media do work for us. Linked lists were crafted to overcome the significant memory management issues that plagued early computing machines whose storage capacity was very limited. Linked lists were one new form of materiality and representation designed to make logical propositions into digital things.

Anthropologist Jack Goody asked - "What's in a list?"[72] He proposed that the evolution of human thinking was tied to the development of literary technologies like tables, charts, and lists. He wanted to know what ways of thinking, what forms of

[71]Where now "data" occupies a central position in our computational vocabulary, the word "information" used to be the catch-all for what was inside computers and what they were representing in the world. The transformation from the language of "information" to the language of "data" hasn't been thoroughly historicized. Relevant works include Thomas Haigh "Inventing Information Systems: The Systems Men and the Computer, 1950 - 1968" in *The Business History Review,* Vol. 75, No. 1 (Spring 2001), pp. 15 - 61; Daniel Rosenberg, "Data before the Fact," in *"Raw Data" Is an Oxymoron*, ed. Lisa Gitelman (Cambridge, MA: The MIT Press, 2013): 15- 39.

[72]Jack Goody, *The Domestication of the Savage Mind* (Cambridge, UK: Cambridge University Press, 1977): 74 - 111.

social organization, and what relationships to temporality and history were made and made possible with the development of written lists. I ask - "What's in a linked list?" Where did that structure come from and what were its consequences for the material history of mathematics? What and how do linked lists represent? To whom or to what?

Linked lists are a window into the kind of work that was involved in putting mathematics into the computer. Part of this story is about the displacement of paper and of the human hand - a displacement of *Principia* and the moment in the material history of mathematical practice that it embodies. However, as we will see in the concluding section of this chapter - the computer seldom replaces paper and it never replaces the human hand. Rather, computing *displaces* paper, *displaces* human labor, directs them towards different goals and problems.

A Mathematical World On Paper

In the section "Proving Theorems in the Twentieth Century" in the Introduction to this dissertation, I introduced Whitehead and Russell's canonical text *Principia Mathematica*. It was part of a particular mathematical tradition that sought to reduce mathematics to logic. Whitehead and Russell aimed to craft formal systems consisting of sets of axioms and rules of inference for deriving consequences from them.[73] Their project reflected a particular philosophy of mathematical knowledge - that it could be fully formalized, axiomatized, and standardized within mathematical logic. For them, a proof consisted of a chain of deductive steps that began with the axioms of their logical system and concluded with a true and interesting logical statement, and they believed the whole of mathematical knowledge could be formulated in this way.

[73]These are laid out in Whitehead, Russell, *Principia Mathematica*, Volume One, pp. 94 - 101.

However, the project also demonstrated a commitment to a particular *medium* as the site where mathematical knowledge should be produced and circulated - namely, paper and the book. Proofs were not just sequences of deductive steps in the abstract, but literally written sequential lists of logical propositions, each step marked by the principle of inference that permitted one to follow from the next. Mathematical knowledge was not hovering in the platonic ether, it was collected, catalogued, and circulated as lists of theorems, lists of inferences, and lists of proofs on the page and collected in the three volumes of *Principia* were first published by Cambridge University Press in 1910, 1912, and 1913.

In order to equip the heads and hands of mathematicians to do the work of proof, on paper, within their logical system, Whitehead and Russell also crafted a particular *notational* system - a way of writing, representing, and exploring logic on the page. Proof would be standardized not just formally but symbolically as well.[74] In their account, the notation was not merely an incidental or arbitrary tool used to access the truths of logic, but a necessary condition for the exploration of logic in the first place:

> The symbolic form of the work has been forced upon us by necessity: without its help we should have been unable to perform the requisite reasoning. It has been developed as the result of actual practice, and is not an excrescence introduced for the mere purpose of exposition. [...] No symbol has been introduced except on the ground of its practical utility for the immediate purposes of our reasoning.[75]

They started with the notational system developed by Peano - an Italian mathematician who worked on the project of providing an axiom system for the natural numbers[76]

[74] A helpful introduction to the notational system in *Principia*, including a comparison with more contemporary logical notation is available in Bernard Linksy, "The Notation in *Principia Mathematica*" in *Stanford Encyclopedia of Philosophy* (Fall 2011): http://plato.stanford.edu/archives/fall2011/entries/pm-notation/.

[75] Whitehead, Russell, *Principia*, viii.

[76] See, for example, Peano, [1889] "The principles of arithmetic, presented by a new method" in Jean van Heijenoort, *A Source Book in Mathematical Logic, 1879–1931.* (Cambridge, MA: Harvard University Press, 1967): 83–97.

- and praised him for showing "how symbolic logic was to be freed from its undue obsession with the forms of ordinary algebra, and thereby made it a suitable instrument for research."[77] Algebra was typically interpreted such that the symbols and variables stand in for unknown *numerical* values or quantities, whereas the symbols of logic in Peano's rendering and Russell and Whitehead's work are taken to stand in for propositions or entities of any kind, making it a more powerful language. Part of the project of the *Principia* was to establish this more powerful potential of symbolic formalism, and to enlarge the scope of algebra-like symbol systems beyond the numerical domain.[78]

In the Introduction to *Principia*, they go to even greater lengths, emphasizing the importance of their notational system for the work of mathematical logic by offering five justifications of their symbol system in relation to the limitations of both natural language and numerically-bounded symbol systems. Of particular interest here are the third and fourth reasons: "The adaptation of the rules of the symbolism to the processes of deduction aids the intuition in regions too abstract for the imagination readily to present to the mind the true relation between the ideas employed."[79] That is to say, the symbol system makes possible more abstract cognition and imagination than natural languages while still capitalizing on the sequential, left to right form of

[77]Whitehead, Russell, viii.

[78]There is an interesting parallel for this shift in history of mathematics from numerical to nonnumerical symbol systems in history of computing. As discussed in "The Possibilities of Computing" in the Introduction, automated theorem-proving practitioners were among those who wanted to use computers to perform nonnumeric tasks, where they were first used primarily for calculation. Both historical moments call for a new way of mobilizing a set of tools - written symbolism in the former and computing machinery in the latter - for new domains. Both begin with a set of tools crafted for numerical work, and fashioned new uses and new interpretations of them that enabled additional domains of application. Both transformations also have ontological stakes concerning what a formal system *is* - be it deductive or algorithmic. Both answer that formal systems are not limited to numerical domains. Natural language, cognition, medial diagnosis, face recognition - all manner of domains - are then opened up to representation and exploration with formal tools like that of algebra or calculation, de-numericized as logic and computation. However, bringing formal tools to bear on these many domains transforms them - they are understood and practiced differently as they are fashioned in terms of deduction or algorithm.

[79]Whitehead, Russell, *Principia*, p. 2.

natural language for readability.

They go on to attribute this potential precisely to the visual properties of the symbol system - which renders abstract ideas and relations concretely and succinctly to the human eye: "The terseness of the symbolism enables a whole proposition to be *represented to the eyesight as one whole*, or at most in two or three parts... This is a humble property, but is in fact very important in connection with the advantages" of the symbol system for the intuition of very abstract objects.[80] The centrality of symbolism in mathematical logic was also emphasized by Peano and Frege before them and David Hilbert and Wilhelm Ackermann after them. Hilbert and Ackermann, for example, wrote that their logic employs "a symbolic language like that which has long been in use to express mathematical relations. [...] The great advances in mathematics since antiquity, for instance, in algebra, have been dependent to a large extent upon success in finding a usable and efficient symbolism."[81]

Principia thus combines a formal system (propositional logic), a medium (paper and the book), and a way of writing mathematics (their notational system) to embody the turn of the twentieth century logicist vision of mathematics. As seen in Figure 1.1 and Figure 1.2, the notational system was both a handwritten part of the production of *Principia* and was reproduced in its typesetting for publication.

On those pages, each line is created by the application of an accepted rule of inference to the proposition on the line before. Each line contains a single proposition with implication, written as $\supset$, as the main logical operator. The left hand implies the right hand. The topmost statement is the theorem to be proved. The proof follows,

[80]Whitehead, Russell, *Principia*, 3, my emphasis. The other justifications can be paraphrased as follows: 1) Natural language is insufficiently abstract and precise for the work of mathematical logic; 2) The grammar of natural language is too flexible for mathematical logic; 5) In order to attain the "complete enumeration of all the ideas and steps in reasoning employed in mathematics," a formal, rigorous symbol system is required. The justifications are enumerated on pp. 2 - 3.

[81]Hilbert, Ackermann, *Principles of Mathematical Logic*, p. 1.

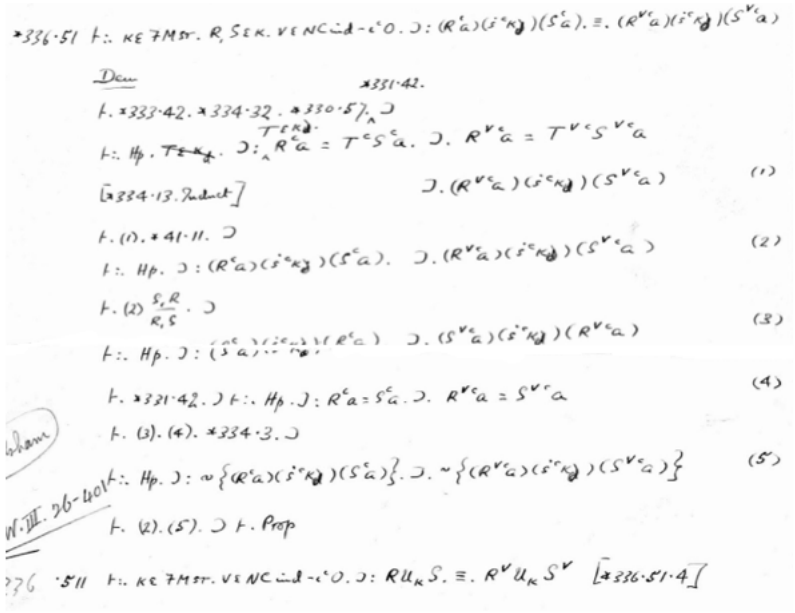


Figure 1.1: Manuscript leaf from *Principia Mathematicia*. MU-BR, c.a. 1907.

***336·51.** $\vdash :. \kappa \in FM\,\mathrm{sr} . R, S \in \kappa . \nu \in \mathrm{NC\,ind} - \iota'0 . \supset :$
$(R^{\iota}a)(\dot{s}^{\iota}\kappa_{\dot{\beta}})(S^{\iota}a) . \equiv . (R^{\nu\iota}a)(\dot{s}^{\iota}\kappa_{\dot{\beta}})(S^{\nu\iota}a)$

Dem.

$\vdash . *333{\cdot}42 . *334{\cdot}32 . *330{\cdot}57 . *331{\cdot}42 . \supset$

$\vdash :. \mathrm{Hp} . \supset : T \in \kappa_{\dot{\beta}} . R^{\iota}a = T^{\iota}S^{\iota}a . \supset . R^{\nu\iota}a = T^{\nu\iota}S^{\nu\iota}a .$

$[*334{\cdot}131] \quad \supset . (R^{\nu\iota}a)(\dot{s}^{\iota}\kappa_{\dot{\beta}})(S^{\nu\iota}a) \qquad (1)$

$\vdash . (1) . *41{\cdot}11 . \supset \vdash :. \mathrm{Hp} . \supset : (R^{\iota}a)(\dot{s}^{\iota}\kappa_{\dot{\beta}})(S^{\iota}a) . \supset . (R^{\nu\iota}a)(\dot{s}^{\iota}\kappa_{\dot{\beta}})(S^{\nu\iota}a) \qquad (2)$

$\vdash . (2)\frac{S, R}{R, S} . \quad \supset \vdash :. \mathrm{Hp} . \supset : (S^{\iota}a)(\dot{s}^{\iota}\kappa_{\dot{\beta}})(R^{\iota}a) . \supset . (S^{\nu\iota}a)(\dot{s}^{\iota}\kappa_{\dot{\beta}})(R^{\nu\iota}a) \qquad (3)$

$\vdash . *331{\cdot}42 . \quad \supset \vdash :. \mathrm{Hp} . \supset : R^{\iota}a = S^{\iota}a . \supset . R^{\nu\iota}a = S^{\nu\iota}a \qquad (4)$

$\vdash . (3) . (4) . *334{\cdot}3 . \supset$

$\vdash :. \mathrm{Hp} . \supset : \sim\{(R^{\iota}a)(\dot{s}^{\iota}\kappa_{\dot{\beta}})(S^{\iota}a)\} . \supset . \sim\{(R^{\nu\iota}a)(\dot{s}^{\iota}\kappa_{\dot{\beta}})(S^{\nu\iota}a)\} \qquad (5)$

$\vdash . (2) . (5) . \supset \vdash . \mathrm{Prop}$

Figure 1.2: Typeset page from *Principia Mathematica*, first edition, 1910.

and is meant to be read from top to bottom. This kind of symbolic system is taken largely for granted today, but it is worth noting that it has a particular *spatiality* and *materiality* and that these have built in assumptions about mathematical agents and mathematical practice.

Principia represents a particular moment in the history of mathematical writing, mathematical materiality, and mathematical practice.[82] Here, the agent of proof is assumed to be a reasoning, seeing, writing, reading person and the notation system is developed accordingly. Material representational systems are tools for thinking, for making, and for communicating mathematical knowledge. The notational system with which Russell and Whitehead make the propositional logic manifest in their work is neither secondary to the project, nor arbitrarily construed. Rather they understood the notation as the enabling condition for constructing and cognizing the deductive system upon which they wanted to ground all of mathematics. It was not possible to transport this written system, unaltered, into the context of computation. A different

[82]The relationship between notational systems and cognition has recently become a subject of interest for science studies scholars. For example, Ursula Klein has explored the role of "paper tools" in the knowledge-practices of nineteenth century organic chemistry in *Experiments, Models, Paper Tools: Cultures of Organic Chemistry in the Nineteenth Century.* (Stanford, CA: Stanford University Press, 2003). Hans Jörg Rheinberger makes a compelling case that way in which scientists take notes during their experimental work shapes the conclusions they will later make concerning what that work was about and what it revealed. See in particular, "Scrips and Scribbles" in *MLN*, Vol. 118, No. 3 (April, 2003): 622 - 636. Bruno Latour suggests that a study of the making, circulation, and reproduction of "inscriptions" would go a long way to understanding knowledge-production in for example, "Visualization and Cognition" in *Knowledge and Society: Studies in the Sociology of Culture Past and Present*, eds. Henrika Kuklick and Elizabeth Long (Jai Press Inc., 1986): 1 - 40. An interest in notational systems has also become an interest for certain historians of mathematics. Notably, the work of Reviel Netz on origin of deductive reasoning practices with the lettered diagram and certain linguistic formulations in ancient Greek mathematics. See, for example, Netz, *The Shaping of Deduction in Greek Mathematics: A Study in Cognitive History* (Cambridge, UK: Cambridge University Press, 1999); Netz, "Linguistic Formulae as Cognitive Tools" in *Pragmatics and Cognition*, Vol. 7, No. 1 (1999): 147 - 176. Rotman, as discussed earlier in Part One, also has a pointed interest in mathematical notation systems in so far as he wants to reduce mathematics to *writing* and associated semiotic practices. See especially Rotman, *Mathematics as Sign: Writing, Imagining, Counting* (Stanford, CA: Stanford University Press, 1993); and Rotman, "Thinking Dia-Grams : Mathematics and Writing" in ed. Mario Biagioli, *Science Studies Reader* (New York: Routledge, 1999): 430 - 441.

notational system and representational structure would be needed if *Principia* was to be automated. It would have to accommodate quite a different mathematical agent, abled and limited in significantly different ways than the imagined readers of *Principia*. I now turn to the development of Herbert Simon, Allen Newell, and John Clifford Shaw's Logic Theory Machine to explore how that was done.

Proof As Information Processing

The Logic Theory Machine was intended to simulate a human mind. The program developed between 1955 and 1958 at the RAND Corporation in Santa Monica, California. The character of the program and the beliefs about minds and computers that were built into it reflect this institution. RAND was founded in 1946 as Project RAND at Douglas Aircraft Company but soon split off to become the non-profit RAND Corporation, distancing itself from product-oriented work.[83] The funding for RAND came from the Air Force who tasked RAND to develop "a robust general science of warfare in which winning strategies and necessary tactics could be derived to achieve global superiority over an aggressor."[84] That is to say, the Air Force wanted to know what future wars would look like and how they should be fought, a difficult task especially in light of uncertainties concerning the ever changing technological landscape of war.[85]

RAND researchers approached this mandate through the study of management, organization, and decision making during war-time especially in the face of uncertain

[83]The emergence and early history of Project RAND and the RAND Corporation is explored in Martin Collins, *Planning for Modern War: RAND and the Air Force 1945 - 1950* (Dissertation submitted to the Department of History, University of Maryland, College Park, MD: 1998); Agatha Hughes and Thomas Hughes, eds., *Systems, Experts, and Computers: The Systems Approach in Management and Engineering, World War II and After* (Cambridge, MA: MIT Press, 2000).

[84]Hounshell, "The Medium is the Message, or How Context Matters: The RAND Corporation Builds an Economics of Innovation, 1946 - 1962" in *Systems, Experts, and Computers*: 255 - 310, p. 259.

[85]This mandate following WWII in which the development of radar, computing, and nuclear weapons had dramatically altered outcomes and experiences of war strategizing.

knowledge. The overall approach to research at RAND was unified under the rubric of "systems analysis" - in which weapons, surveillance and intelligence equipment, committees, political organizations, and other technological and human components were understood as holistic and dynamic systems that could be mapped and optimized.[86] Systems engineering was clearly informed and transformed by the emergence of cybernetics during World War II in which vocabulary and tools for studying certain humans, animals and machines as self-regulating, dynamic systems was formalized. At RAND, the systems engineering approach was applied to military strategy, economics, Soviet Studies, and to the human mind and the idea of artificial minds.

Newell left a graduate program in mathematics at Princeton in 1950 and came to RAND as a consultant. Once there, he was inducted into the systems engineering approach to problem solving almost immediately, being tasked upon arrival with participation in a then-ongoing project to model and understand the American air-defense system as manifested in McChord Field Air Defense Direction Center in Tacoma. This

[86]The systems approaches to engineering have their origins in Word War II. While no single set of properties unified the approach into a single whole, a common feature was the treatment of technologies, people, and organizations of both as systems with common properties. This approach of integrating human and machine elements as parts of hybrid systems bears the mark of cybernetics - the study of organic and inorganic entities according to the same interests of feedback and self-regulation. Peter Galison has explored the origins of cybernetics in the context of WWII era anti-aircraft gunnery development in (Galison, "The Ontology of the Enemy" in *Critical Inquiry*, Vol. 21, No. 1 (1994): 228 - 266). Roberto Cordeschi also provides an account of the development of cybernetics within a longer history of interest and concern in studies of teleology in organic and inorganic systems in *Discovery o the Artificial: Behavior, Mind and Machines Before and Beyond Cybernetics* (Dortrecht, Kluwer: 2002), esp. pp. 153 - 186. The systems approach to engineering problems was enabled by the cybernetic proposition that humans, animals, and machines are regulated by the same processes and can therefore be integrated into holistic systems. David Mindell identifies a changing notion of "system" emerging during World War II in particular in the development of radar technologies. The Rad Lab at MIT, in particular, engaged with a notion of system as a "dynamic entity" rather than a stable sum of constituent parts. See Mindell, "Automation's Finest Hour: Radar and System Integration in World War II" in *Systems, Experts, and Computers*, pp. 27 - 56. Ideas of system, cybernetics, and the context of WWII are also the subject of Paul Edwards, *The Closed World: Computers and the Politics of Discourse in Cold War America* (Cambridge, MA: MIT Press, 1996), esp. Chapter One ""We Defend Every Place": Building the Cold War World" (pp. 1- 42) and Chapter Four "From Operations Research to the Electronic Battlefield" (pp. 113 - 146). This new idea of a system expanded with cybernetic notions of feedback and self regulation and through these related trajectories, especially in the hands of RAND consultants and employees, systems engineering came into being.

project also afforded Newell and Simon their first meeting when the latter began consulting for the project in 1952.

Simon arrived with an interest in human decision making, in organizations, and in reason already in mind from his previous work.[87] As an economist, psychologist, and business management researcher, Simon was invested in processes of decision making and judgement in individuals and organizations. His dissertation project of the late 1930s, for example, came out of the application of logic to the study of administrative behavior - a work for which he eventually won the Nobel prize in economics in 1978. The air-defense experiments also landed Newell in the RAND basement where he encountered John Clifford Shaw, who would become the lead computer practitioner who did much of the work *implementing* the Logic Theory Machine. Shaw was then working in the numerical analysis department (later to be called the "programming department") providing various computational services to other departments and projects at RAND.

The crux of the air-defense experiments - based in RAND's Systems Research Laboratory (SRL) - focused on understanding human-machine interaction and human decision making processes in the air defense command and control systems. The project studied how the human elements in the system made decisions to, e.g. scramble anti-aircraft planes based on technological information from, e.g. radar. The ultimate goal was to understand this systemic behavior, to model it, and then to optimize it through the development of specialized training programs.[88]

[87] At that time, he held a faculty position at Carnegie Technical Institute, soon to become Carnegie Mellon University. He would be instrumental in the creation of the latter's department of Computer Science and robust artificial intelligence research community during the 1960s.

[88] Newell produced one RAND report related to this project: "Description of the Air-Defense Experiments II. The Task Environment."Newell, Description of the Air-Defense Experiments, II. The Task Environment" RAND publication No. P-659 (October 17, 1955). Simon also wrote one primary RAND report on the subject: Simon, "Observations and Comments on the Organization Studies of the Systems Research Laboratory" RAND publication No. RM-922 (1952). Newell's thinking about systems even precedes this project somewhat. While still at Princeton, Newell had already begun to

By the time Newell, Simon, and Shaw began work on the Logic Theory Machine in 1955, they had been working for several years on projects like this one - thinking about human decision making processes, man-machine systems, and the nature of systems in general.[89] In the systems approach, learning, deciding, reasoning, and judging were attributable to man, to man-machine organizations, and eventually to machines themselves. All were understood as "species of the genus information processor" - they took symbolic information as input and manipulated it in order to solve problems, formulate decisions, and otherwise navigate the world.[90]

Newell, Simon, and Shaw's conception of *information* and of *symbols* drew together rich and complex historical trajectories that become increasingly tied to technology through the twentieth century. Information in the twentieth century is perhaps most readily identified with the theory that Claude Shannon developed in the 1940s.[91] Shannon was interested in the study of information in itself, independent of specific content

move away from pure mathematics, taking up a research assistantship with German-born economist Oskar Morgenstern (of game-theory fame). Morgenstern was, at that time, interested in using logic to model rail-yards as systems. Although Newell claims to have found that work uninteresting, his earliest RAND report documented that subject "The Capacity of a Railroad Freight Yard (A Survey of the Problem – Not the Solution)" RAND Document, No. RM-555 (June 14, 1950). Here, we can see evidence of early systems thinking in Newell's work, but with a somewhat more stochastic and game-theoretic methodology than in this work on air-defense in the early years at RAND. For other discussions of the project, see Edwards, *The Cold World*, esp. pp. 121 - 124; McCorduck, *Machines Who Think*, esp. pp. 139 - 145; and Hounshell, "The Medium is the Message, or How Context Matters: The RAND Corporation Builds an Economic Innovation". Newell was the most actively involved in the project and Simon acted as a sometimes visiting consultant. Shaw (who also arrived at RAND in 1950), was then working in the numerical analysis department who provided computational services to other labs and groups at the corporation. Shaw recalls having first met Newell in conjunction with the air defense experiments for which he provided assistance creating the necessary simulations. See McCorduck, J. C. Shaw Interview, June 16, 1975, (CMU_PM, Series III, Transcripts).

[89]Early signs of a more general interest in learning and reasoning - in humans, but in *other systems as well* - are clearly visible there. For example, in reporting the progress of the air defense experiment task environment, Newell wrote that "After the first few sessions the problems were not difficuLogic Theory Machine at all. In fact, *the organization learned its way* right out of the experiment." Newell, "Description of the Air-Defense Experiments", p. 1, my emphasis. His language reveals a belief that the organization could learn, not merely the individuals within it.

[90]Allen Newell, Herbert Simon, *Human Problem Solving* (Upper Saddle River, NJ: Prentice Hall, 1972).

[91]His work culminated with the foundational paper "A Mathematical Theory of Communication" in *Bell Systems Technical Journal*, Vol. 27 (October 1948): 379 - 423.

or semantics. He was also interested in understanding information as a physical thing - transmitted in signals, stored, compressed, moved, etc. - whose limits and behaviors could be modeled and quantified.[92] In both regards, Newell and Simon's notion of "information" is influenced by Shannon. In fact, some of their earliest work developing a programming language to realize the Logic Theory Machine was presented at a July 1956 *Symposium on Information Theory.* Indeed, Shannon's information theory underlay the development of cybernetics and, in turn, systems engineering that so dominated the Systems Research Laboratory at RAND within which Newell and Simon were working.[93]

However, while Shannon's information theory focuses on syntax it is not explicitly a theory of symbols or of formal logic - both of which dominate the character of information in Newell and Simon's work. Crowther-Heyck suggests that Newell and Simon's experience with logic predisposed them "to conceive of decision-making as a logical process of drawing conclusions based on certain premises."[94] They drew from that turn of the century tradition in which human reasoning was associated with deductive logic, itself deeply tied (as we have seen) to the development of particular symbol systems. Their vision of information processing drew together those two traditions - information theory and mathematical logic.

In attributing the same form of symbolic information processing to the computer and the mind, Newell and Simon drew from post-cybernetic resources in which humans, non-human animals, and certain machines were understood as examples of the same

[92]A history of the origins Shannon's information theory can be found in Samuel Thomsen, "Some evidence concerning the genesis of Claude Shannon's information theory" in *Studies in the History and Philosophy of Science* Vol. 40 (2009): 81 - 91.

[93]Hunter Crowther-Heyck offers a more in depth survey of the influence of Shannon's information theory on Newell and Simon and on systems engineering in both "Defining the Computer: Herbert Simon and the Bureaucratic Mind – Part I" in *IEEE Annals of the History of Computing,* Vol. 30, No. 2 (2008): 42 - 51 and "Defining the Computer: Herbert Simon and the Bureaucratic Mind – Part II" in *IEEE Annals of the History of Computing,* Vol. 30, No. 2 (2008): 52 - 63.

[94]Crowther-Heyck, *Herbert A. Simon*, p. 219.

kind of thing - self-guided or goal-directed systems, governed by the same sets of rules. Even before Wiener, Rosenblueth, and Bigelow published their famous paper "Behavior, Purpose, and Teleology," psychologists, behavioral scientists, and even some engineers were theorizing, exploring, and experimenting with the idea of purposive and intentional behavior. They asked what mechanisms and rules enable feedback with the environment and engender purposive behavior?[95]

One answer to this question came from psychologist Kenneth Craik. According to Cordeshi, his work had a visible and traceable impact on those working in early artificial intelligence, including Newell and Simon. Craik proposed that purposive behavior was made possible when an entity (man, animal, or machine) could generate a *symbolic* representation of its environment.[96] Within Craik's "symbolic theory of thought," intentional agents would engage with their model of the environment through feedback processes that would enable and shape intentional behavior. It was through a feedback processes within a symbolic representation of the world that Craik explained human, some animal, and automata purposive behavior without theorizing teleologically.

Newell and Simon were among the early AI practitioners that identified Craik as a guiding influence, and Craik's theory serves to illuminate the emphasis on *symbol systems* in their work. "Information processing" for Newell and Simon mobilized the idea of signals from Shannon, the notion of reasoning as logical deduction, and the idea that symbolic representations of the world can enable intentional behavior. Their idea of a system was informed by systems engineering, cybernetics, and operations research

[95]Concern with goal-directed behavior was wide spread in Europe and the United States throughout the twentieth century, before the emergence of cybernetics proper just before the second word war. Purposive behavior was taken to include those actions that seemed to be in service of some future outcome, and it became a concerning phenomenon as scientists became increasingly distrustful of "teleological" explanations. Corseschi suggests, in *Discovery of the Artificial*, that the search for laws and mechanisms that could explain purposive behavior and "explain away" its seemingly teleological character were long sought among behavioral scientists and psychologists throughout the century. See, especially "Chapter Four: Behavior, Purpose and Teleology."

[96]Cordeschi, *Discovery of the Artificial*, p. 141 - 142.

that were deployed everywhere around them.

All of these influences are present in the notion of a "complex information processing system" - a notion that included the Logic Theory Machine and that the Logic Theory Machine was meant to illuminate further through empirical research. And for Newell and Simon, human reasoning was also a complex information processing system (though manifested in a different physical system). By way of that similarity, Newell and Simon famously believed that the Logic Theory Machine, once in operation, was in fact a simulation of human thinking because for them, that faculty was precisely the manipulation of symbolic information:

> When we say that these programs are simulations of human problem solving, we do not mean merely that they solve problems that had previously been solved only by humans - although they do that also. We mean that they solve these problems by using techniques and processes that resemble more or less closely the techniques and processes used by humans.[97]

In fact, Newell and Simon went so far as to say that the best way to understand human behavior was in terms of a computer program that specified the rules underlying the information processing at work in it:[98]

> We wish to emphasize that we are not using the computer as a crude analogy to human behavior. We are not comparing computers with brains, nor electrical relays with synapses. Our position is that the appropriate way to describe a piece of problem-solving behavior is in terms of a program: [...] in terms of certain elementary information processes it is capable of performing. [. . .] Digital computers come into the picture only because they can, by appropriate programming, be induced to execute the same

[97]Newell, Shaw, Simon, "The Processes of Creative Thinking," RAND Corporation Technical Report P-1320 (1959), p. 6. Here they also document some of their later work developing chess-playing programs and a so-called "General Problem Solver."

[98]The notion that all human reasoning was *rule-bound* was of course tied in part to turn of the century efforts to formulate reason as a logical deductive system consisting of axioms and inference rules. For discussion of where this rule-bound conception of human reasoning came from and its significance for postwar theories of mind, see Paul Erickson, et al, *How Reason Almost Lost its Mind* (Chicago, IL: University of Chicago Press, 2013): esp. pp. 27 – 51.

> sequences of information processes that humans execute when they are solving problems.[99]

Newell and Simon explicitly situated the Logic Theory Machine project in the broader RAND goal of studying complex systems: "One tactic for exploring the domain of complex systems is to synthesize some and study their structure and behavior empirically. This paper provides an explicit specification for a particular complex information processing system - a system that is capable of discovering proofs for theorems in elementary symbolic logic."[100] The systems approach enabled them to talk about reasoning as a property of certain kinds of systems; cybernetics enabled them to identify the computer as such a system; and RAND air defense research afforded them both the opportunity to meet one another and provided the resources required to pursue the Logic Theory Machine.

Although RAND was funded by the Air Force and had an overarching mandate to prepare that organization for future wars, the research environment was, by nearly all counts, quite flexible and liberal.[101] Indeed, independent research was encouraged and expected of its employees and consultants. Throughout the 1950s, RAND moved away from research with immediate relevance for military strategy and towards a more general mantra of applying systems engineering to important problem domains. In fact, RAND historian Martin Collins suggests that the philosophy of independent research was as prominent as the sense of urgency in "fighting the Russians" that was behind RAND and other so-called Cold War institutions' research.[102]

[99]Newell, Shaw, Simon, "Elements of a Theory of Human Problem-Solving" in *Psychological Review*, Vol. 65, No. 3 (1958): 151 – 166. For a more in depth exploration, see Crowther-Heyck, Herbert A. Simon, esp. "A New Model of Mind and Machine" (pp. 184 – 214) and "The Program Is the Theory" (pp. 215 – 232).

[100]Newell, Simon, "The Logic Theory Machine: A Complex Information Processing System", p. 3.

[101]See, for example, Paul Armer's account, cited in McCorduck, *Machines Who Think*, p. 139.

[102]Martin Collins, *Planning for Modern War: RAND and the Air Force, 1945 - 1950* (PhD dissertation submitted to University of Maryland, History Department: 1998), p. 2.

This position was defended on the grounds that, because of the expansive and scientific character of warfare in the twentieth century, "modern war, more than in previous periods, was a contest between whole societies, not just between opposing military forces".[103] The mobilization of technology and knowledge required to win such wars could not be directed by military expertise alone. Although the overall RAND approach to research was characterized by an basic interest in military systems management and was deeply characterized by ties by cybernetics and weapons development, RAND employees and consultants were free to work on projects without immediate defense applications.

The Logic Theory Machine is an exemplar of this kind of Cold War research project - its character is inextricable from the systems engineering methodology of 1950s RAND research which is, in turn, deeply tied to fear of a Soviet threat, World War II era weapons design, and future armed conflict strategizing - but it was not directly or easily applicable to immediate weapons design or war planning. In this sense, the Logic Theory machine echoes what Hounshell describes as the "new frontier" of problems and concerns offered up by the Air Force, but enabling and engendering myriad different projects and research trajectories.

The Logic Theory Machine was made possible by the Cold War. Not only were the required institutional and financial resources a product of the increasingly integrated military-academic-industrial complex. The kinds of questions being asked and problems being solved by such Cold War institutions made possible new ways of thinking about cognition, humans, machines, decisions, and so on that made it possible to imagine a machine that could prove mathematical theorems. In this sense, the Logic Theory Machine is an example of the simultaneous constraining and enabling influence of Cold War concerns and intellectual focuses.

[103]Collins, *Planning for Modern War*, p. 2.

The program was an experiment within a particular research methodology aimed at providing opportunities for empirically investigating a rather abstract and multi-faceted concept - a "complex system."[104] It was, in a sense, of the same *kind* as the air defense system simulation experiments pursued at RAND earlier in the 1950s - an attempt to construct a complex system, and then to study its behavior in pursuit of better models and understanding of what complex systems are. It is also the case that a certain model of reasoning was already *built in* to the program, namely, a theory of reasoning-as-information-processing - but this theory was informed by the broader experimental research context of systems engineering in air defense.

Trick or Tree: A Heuristic Theorem Prover

Newell and Simon approached the prospect of automated theorem-proving with an interest in automating what human mathematicians do when they prove theorems. Their interest was not in finding new ways to prove theorems or in the improvement of mathematical research prospects by way of digital computing but rather in bringing computers up to the task of what human mathematicians already do. Their interest in mathematics was as a window into the mind - a paradigmatic example of human reasoning to be mapped and understood in the service of a broader effort to model the mind.[105]

[104]Newell and Simon define their meaning of "complexity" relative to systems engineering in "The Logic Theory Machine: A Complex Information Processing System", on p. 2.

[105]Logic was not the first problem domain to which Newell and Simon turned with their interest in automating human reasoning processes. Newell and Simon also worked extensively on the automation of chess playing. In the summer of 1954, Oliver Selfridge visited RAND and presented his work on developing so-called "pattern recognition skills" for simple alphabets. See, for example, Selfridge, Dineen, "Pattern recognition and modern computers" in *Proceedings of the 1955 Western Joint Computer Conference* (March 1 - 3, 1955): 91 - 93. Newell reported in an interview with Pamela McCorduck in 1974 that Selfridge's work led to his initial revelation that the computer could be treated as a system for manipulating symbolic information. (McCorduck interview with Allen Newell, September 30, 1974. From CMU-PM: Series II: http://doi.library.cmu.edu/10.1184/pmc/newell/box00088/fld06028/bdl0002/doc0001, p. 18). In-

They were in the business of theorizing reasoning *in general* as a property of certain kinds of systems that was thought to include both the human mind and the modern digital computer. Newell and Simon's posture toward the project of automation of logic as a *means* and not an *end* differs from the position adopted by many automated theorem-proving researchers who followed. Many such later researchers hoped their theorem-proving programs would make contributions to mathematics and/or transform the way that mathematics was done.

One explanation for Newell and Simon's relative disinterest in mathematical research for its own sake is the fact that, unlike many who followed, Newell and Simon were not mathematicians.[106] Neither was a member of a mathematical community, nor prioritized mathematical knowledge in itself in their work. Both men received advanced training in mathematics and in logic and at times in their lives pursued training

spired by Selfridge's program, Newell designed a program that would play chess - but not by checking every possible move. See Newell, "The Chess Machine: An Example of Dealing with a Complex Task by Adaptation" in *Proceedings of the 1955 Western Joint Computer Conference* (March 1 - 3, 1955): 101 - 108. Newell's choice of chess as a problem domain may have resulted from the fact that he himself played, or it may be because of the long history of the automation of chess-playing that preceded his work. For an account of the centrality of chess to research on the automation of reasoning see Nathan Ensmenger, "Is Chess the Drosophila of Artificial Intelligence? A Social History of an Algorithm" in *Social Studies of Science*, Vol. 42, No. 1 (2012): 5 - 30. Chess, however, created a lot of practical problems given the huge number of possible moves that can be made at any point, and the wide range of different "goals" that can prescribe choices between moves (e.g., checking, check mating, taking the opponent pieces, taking the queen, etc.). During the Fall of 1955, when Newell was working at RAND and also pursuing graduate studies with Simon at Carnegie Mellon, the two met every Saturday to discuss chess problems but they also brainstormed other problem domains that might be more tractable for a full-scale automation attempt. (This account comes from a memoir Simon drafted in 1957 when reflecting on 1954 and 1955. The account is excerpted in McCorduck, *Machines Who Think*, pp. 162 - 163). In these conversations they quickly considered theorem proving. Newell and Simon first wanted to design a theorem prover for plane geometry, but eventually abandoned this project given the centrality of *diagrams* in human pursuits of this field - they imagined this would be difficult to automate. They then came to *Principia*'s propositional logic as a domain of human reasoning, tractable for the project of automation. Simon suggests that they chose *Principia* simply because he "had the *Principia* of Whitehead and Russell at home, and [he] pulled it off the shelf one day to have some problems." (McCorduck, interview with Herbert Simon, as cited in *Machines who Think*, p. 161).

[106]Donald MacKenzie suggests a similar possibility when overviewing different approaches to automated theorem proving in "The automation of proof: A historical and sociological exploration" in *IEEE Annals of the History of Computing*, Vol. 17, No. 3 (1995): 7 - 29.

in pure mathematics - but each decided that mathematics for its own sake was not for them.

Simon was an undergraduate student at the University of Chicago, majoring in political science, with concerted interest in economics, business, and psychology.[107] His interest in logic and mathematics stemmed from the perceived power garnered by applying its tools to economics, studies of government, and psychology.[108] For example, while at the University of Chicago, Simon took Rudolf Carnap's advanced research seminar on logic, but his final paper offered an account of the "Logical Structure of a Science of Administration" - a project that later became his PhD dissertation in economics.[109]

Simon eventually distanced himself from formal mathematics training, turning instead to self-instruction. In his autobiography, he depicts his undergraduate self as having been adequately trained in high school to skip most of his classes, choosing instead to audit higher level courses from an early age.[110] This approach to college education eventually contributed to his distance from mathematics training - "Early in my second year, I terminated my formal education in mathematics when a calculus professor insisted that I attend class. From then on, almost all of my knowledge of mathematics was self-taught, some of it while I was at the university, but continuing fairly intensively until the early 1950s, carrying through most of the subjects in a doc-

[107] A comprehensive account of Simon's university career at the University of Chicago is available in Crowther-Heyck, "Chapter Two: The Chicago School and the Sciences of Control" in *Herbert Simon,* pp. 31 - 59. Simon's autobiographical account of his university experiences can be found in "Education in Chicago" and "Encounter with a Scientific Revolution: Political Science at Chicago" in *Models of My Life*, pp. 36 - 68.

[108] Simon attributes his belief in the power of rigorous mathematical analysis for theory-building and modeling in economics especially to the influence of University of Chicago-based economist Henry Simons. See Simon, *Models of My Life*, p. 39.

[109] CMU-HS: Series II: Schoolwork and Early Career: Abstract: University of Chicago – Research Seminar in Logic Papers and Rough Drafts, dated July 27. 1937 http://doi.library.cmu.edu/10.1184/pmc/simon/box00002/fld00124/bdl0006/doc0002.

[110] Simon, *Models of My Life*, p. 39.

toral curriculum of that time (lots of higher algebra, analysis, and function theory; little topology). Self-instruction gave me the courage and skill to master new areas of mathematics whenever I needed them for my research. It also left me with mathematical skills that are more rough-and-ready than polished."[111] Simon certainly had respect for mathematics, and extensive knowledge of it, but was not part of a mathematical community and did not aim to make contributions to mathematical knowledge for its own sake.

Newell majored in physics as an undergraduate at Stanford University but later began a graduate degree in mathematics at Princeton University. However, he too found a reason to defect. In an interview with popular AI historian Pamela McCorduck in the 1970s, Newell described discovering the following about himself while studying at Princeton:

> I was a problem solver, and I wanted problems you could go out and solve. I simply couldn't understand what motivated pure mathematicians to go on working, looking at the structure of some mathematical logic. So Princeton and I just passed in the night. They sort of acted like they'd be glad to have me back when I left at the end of the year, but what I'd found out was that I could't have led that life for anything. None of their concerns are my concerns, and I learned that, and it took me a year to learn it, and then I got out of mathematics.[112]

It was after that first year of graduate work that Newell accepted a summer internship at RAND - and there he got his wish of going out and solving problems in spades, being put to work immediately on practical research for the Air Force.

While anecdotal, Newell and Simon's narration of their relationship to mathematics reveal a clear characterization of the discipline as a *tool* for solving other problems, and also as a means through which the reasoning of the human mind could be recognized and theorized. For Newell and Simon, mathematics was itself an example domain of

[111] Simon, *Models of My Life*, p. 40.
[112] McCorduck, Allen Newell Interview, as cited in *Machines Who Think*, p. 140.

a larger phenomenon that was their ultimate target - human reasoning. They wanted to automate theorem-proving as an exercise in identifying the symbolic processing behaviors at work in human reasoning rather than a contribution to mathematical research for its own sake. They wanted to design a computer program in which would be codified human mathematical reasoning.

Before they could continue with the design of their program, they needed an account of what reasoning in the domain of logical problem-solving looked like. To this end, Newell and Simon's work on the Logic Theory Machine was informed by one mathematician in particular: Hungarian George Polya.[113] In addition to his influential mathematical research, Polya had a significant interest in mathematics pedagogy. He believed, contrary to many myths and idealizations of mathematical work, including that presented in *Principia*, that mathematicians don't just go about deducing conclusions from axioms in hopes of inferring true and interesting conclusions. When mathematicians set out to prove a conjecture they do all kinds of things. They experiment with many cases looking for patterns, they develop analogies with other problems, they try to formulate related mathematical objects, they work backwards from something they already believe is true, they look for counter-examples, and so on. It can be a long way from axioms to desired conclusions by deduction, and mathematicians look for short-cuts to help them find a proof by other means. As powerful as the modern project of axiomatizing modern mathematics may have been, it did not embody *what mathematicians actually do.*

Polya set about to identify and articulate the actual tricks and practices of mathematicians in a two volume work in which he wrote "certainly let us learn proving,"

[113]Polya was born in Budapest, Hungary in 1887, leaving an academic post in Zurich in 1940 to emigrate to the United States with so many other Jewish academics. He visited at Princeton and eventually took a post at Stanford which he held until his death in 1985. See Gerald Alexanderson, *The Random Walks of George Polya* (The Mathematical Association of America, 2000).

- that is, let us learn how to deduce conclusions from axioms - "but also *let us learn guessing*."[114] He believed that the "ways of guessing" employed by research mathematicians in their actual practices were not esoteric, tacit secrets of the trade but rather could be articulated and formalized as rules. He thought they should be taught to students of mathematics so that they would be better equipped for mathematical research. He called them *heuristics* - from the Greek word *heuriskein* for "find". Central among Polya's heuristics was "reasoning by analogy" and also "induction".[115] Here was an explicit, *rule-bound* model of what human theorem-proving practices looked like - precisely the kind of thing Newell and Simon were after.

Newell started out his freshman year in 1946 at Stanford by taking Polya's course "How to Solve It" - named after the book published just the year before. He was so taken with Polya that he then took what he believes to be every course that Polya offered at Stanford before the end of of that year.[116] While Newell characterized himself

[114]George Polya, *Induction and Analogy in Mathematics* Vol. I of *Mathematics and Plausible Reasoning* (Princeton, NJ: Princeton University Press, 1954), p. vi, emphasis in original.

[115]Polya's two volume work focused on what he called "plausible reasoning" as opposed to *certain* reasoning by identifying ways that mathematicians convince themselves that something is probably true. Volume I of that work focuses in large part on reasoning by analogy, and by empirical deduction from attempted cases in mathematics. In Volume II he devotes a lot of time to pattern recognition. Throughout, he aims to identify rules that govern what kinds of plausible reasoning practices engender what kind and relatively how much insight into the problem at hand. Polya, *Induction and Analogy in Mathematics*, Vol. I of *Mathematics and Plausible Reasoning* and, *Patterns of Plausible Inference*, Vol. II of *Mathematics and Plausible Reasoning* (Princeton, NJ: Princeton University Press, 1954). Before these extensive volumes, Polya had published a more introductory text on mathematical methods and heuristics, called *How to Solve It:A New Aspect of Mathematical Method* (Princeton, NJ: Princeton University Press, 1945) oriented more towards *teaching* mathematics, and including many sample heuristics with examples, and sample problems with solutions. Two other works embody Polya's work on pedagogy and heuristics: *Mathematical Methods in Science* (Washington, DC: The Mathematical Association of America, 1977); *Mathematical Discovery: On Understanding, Learning and Teaching Problem Solving,* Vol. 1 (John Wiley & Sons, 1962); *Mathematical Discovery: On Understanding, Learning, and Teaching Problem Solving*, Vol. 2 (John Wiley & Sons, 1965).

[116]Newell recounted his experiences in Polya's courses in a talk he gave at the International Symposium on the Methods of Heuristic (University of Bern, Switzerland September 15 - 18, 1980): "The Heuristic of George Polya and its Relation to Artificial Intelligence". A preprint of the talk for circulation and comments, dated September 1980, is available from CMU-AN, and a later finalized version from 1981 is available at http://repository.cmu.edu/cgi/viewcontent.cgi?article=3446&context=compsci.

as uninterested in mathematics he admits that he was very interested in Polya. Newell and Simon wanted a computer to do what human mathematicians do, and Polya offered an account of what that was.

If students of mathematics could be taught to prove theorems by following explicit heuristic rules, why couldn't a computer also be? Heuristics would avoid the practical problems of exhaustive instantiation, as discussed in the Introduction to the dissertation, and reflect actual mathematical practice. Newell and Simon adopted the general position that human mathematicians proceed in theorem-proving by heuristic means and that heuristics could be formalized and articulated as rules. If they could be articulated as rules, they could be automated. Heuristic rules constituted the "symbolic information processing" model of proof that they were after.

There was one heuristic in particular - outlined in *How to Solve It* and possibly introduced to Newell in 1946 when he took the eponymous course - that features in the design of the Logic Theory Machine, and that in one sense characterizes its behavior. The heuristic is called "Working Backwards" by Polya, who identifies its origins in Pappus (c.a. 290 - 350 C.E.) and in experimental psychological studies of problem solving behaviors in animals.[117]

Deductive methodology begins from axioms and known theorems and attempts to deduce a desired conclusion from them. In this heuristic method, instead, one begins with the end - starting from the thing to be proved and tries to "work backwards" to the axioms. Citing Pappus, Polya identifies the roots for this heuristic in antiquity: "Let us start *start from what is required* and *assume what is sought is already found*...

[117]See Polya, *How to Solve it*, p. 225. Pagination here is from the Expanded Princeton Science Library Edition (2004). Polya often used empirical examples of his heuristics, sometimes drawn from the experimental study of the natural world, in order to illuminate and engender intuitions for his heuristic methods. In this case he cites an example of trying to draw up a specific volume of river water using two vessels of given size, and demonstrated that (literally) working backwards results in a simple solution (pp. 226 - 230).

Let us *inquire from what antecedents the desired result could be derived.*"[118] In order to prove a proposition, begin by assuming it to be true and then look for antecedent propositions that, if true, lead in one derivative step to the desired conclusion. If such antecedents could be constructed in a sequence that eventually led back to the axioms, then that sequence (run backwards) would constitute a deductive proof of the proposition from which the process began.

The Logic Theory Machine was designed to work backwards in this way. The program included an encoding of the central axioms of Russell and Whitehead's *Principia*, its logical operators, and its rules of inference - substitution, replacement, and detachment.[119] However, it was not programmed to apply the rules of inference to the axioms, seeking the theorems from *Principia* by brute force deduction. Instead, in most cases, the Logic Theory Machine would begin with a *Principia* theorem, and construct a set of logical propositions that lead to that theorem in one permissible inference step. Then, the Logic Theory Machine would construct another set of propositions that, if true, would lead to those propositions in one permissible inference step, and so on. The hope was the eventually, the axioms themselves would be produced as subproblems. In that case, the chain of subproblems could be run in reverse, and it would constitute a deductive proof of the theorem from the axioms. In some cases, this method would exhaust the JOHNNIAC's resources before a proof was found, and in other cases, it would be impossible to find a proof this way.[120]

[118]Cited in Polya, *How to Solve It*, p. 227, emphasis in original.

[119]See "Proving Theorems in the Twentieth Century" in the Introduction to the dissertation for an explanation and survey of these elements of *Principia*.

[120]As with most heuristic methods, there was no guarantee that a proof would be found this way. Indeed, that is why Polya was interested in *plausible* reasoning rather than reasoning with certainty. However, humans are not guaranteed to discover a proof by any non-mechanical means of search either, so this was not considered to be a drawback of the program. Neither was their much risk of Logic Theory Machine running forever, or even for very long, without finding a proof. The JOHNNIAC computer on which it was implemented had very limited memory, and was also used by many other practitioners for other purposes. As such, in the absence of a timely proof, the Logic Theory Machine would stop or be stopped for practical reasons. The JOHNNIAC computer and its contribution to

This "Working Backwards" heuristic was only one application of the more basic heuristic method that Newell, Shaw, and Simon called "subproblem generation" - the construction of logical propositions that were connected to other propositions by way of one permitted rule of inference. Subproblems could be generated for any proposition forwards or backwards, and this method was called in general "chaining":

> These methods use the transitivity of the relation of implication to create a new subproblem which, if solved, will provide a proof of the problem expression. Thus, if the problem expression is "a implies c," the method of forward chaining searches for an axioms or theorem of the form "a implies b." If one is found, "b implies c" is set up as a new subproblem. Chaining backward works analogously: it seeks a theorem of the form "b implies c," and if one is found, "a implies b" is set up as a new subproblem.[121]

The particular case of backwards chaining by the production of sets of subproblems for the theorem to be proved and seeking a chained path to the axioms was the most powerful mechanism of the program and was deployed often in Logic Theory Machine runs.

The development of this method of subproblem chaining led to the development of some new techniques for thinking about and representing proof. In order to articulate this method, Newell, Simon, and Shaw represented logical proofs as *trees*, where generated subproblems are child nodes and the originary node is the theorem to be proved.[122] For example, they included the diagram in Figure 1.3 in an early publication on the

characterizing the Logic Theory Machine will be the subject of later sections.

[121]Newell, Shaw, Simon, "Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristic" in *Proceedings of the Western Joint Computer Conference* (1957): 218 - 230, on p. 222.

[122]It is difficult to discern where they took this notion of a tree from. Certainly, tree structures had been deployed in logic in the past, and they were incredibly common in anthropological studies of kinship and in linguistics. Branching tree structures became completely prolific in computing research as early as the 1950s but the exact origins are hard to pinpoint. For a historical account of earlier uses of tree diagrams in logic, see Ian Hacking "Trees of Logic, Trees of Porphyry" in *Advancements of Learning: Essays in Honor of Palo Rossi* [Vol. 62 of *Biblioteca di Nuncis*] (L.S. Olschki, 2007): pp. 219 - 261. Trees are now used prolifically in computing literature to describe, especially iterative, algorithmic processes.

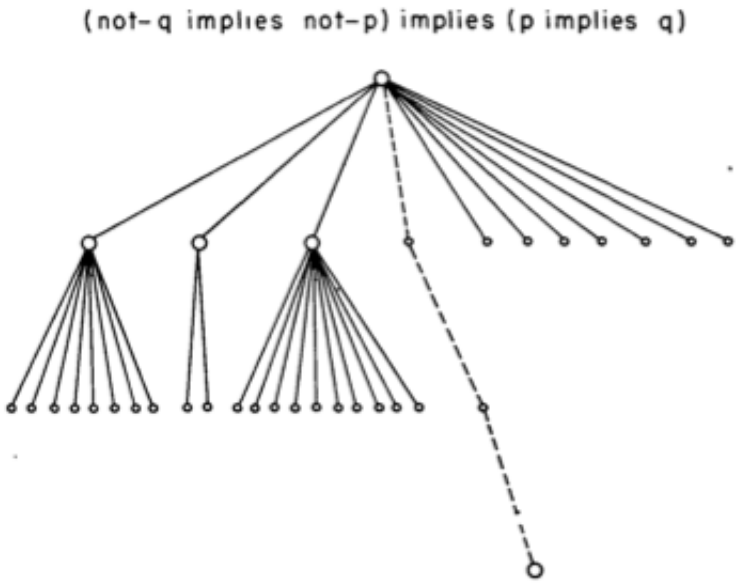


Fig. 7—Subproblem tree of proof by LT of 2.17 (all previous theorems available).

Figure 1.3: Logic Theory Machine Subproblem Proof Tree for Proposition 2.17 from *Principia Mathematica*. Newell, Shaw, Simon, "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristics," p. 228.

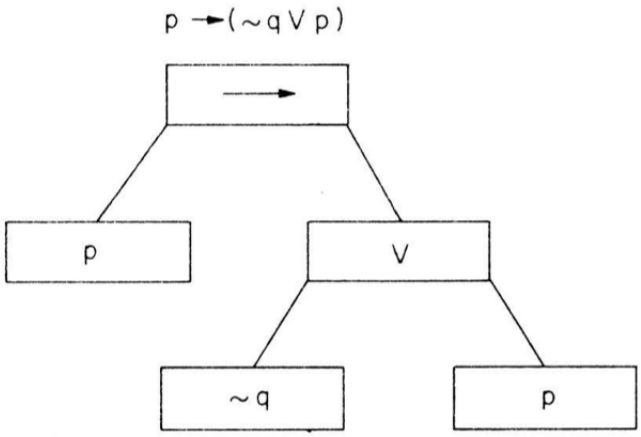


Figure 2. Logic Expression

Figure 1.4: Tree Diagram of a Logical Expression. Newell, Shaw, "Programming the Logic Theory Machine," p. 10b.

Logic Theory Machine to represent the program's search for a proof.[123] At some point in 1955, Newell and Simon turned to the exploration of individual logical propositions *themselves* as trees in which the logical operators that connect literals are the nodes of the tree, and the connected literals are child nodes. For example, Figure 1.4 is a diagram of a proposition-tree as published in 1957 and Figure 1.5 shows Simon's hand drawn tree formulation of many propositions from *Principia* Chapter 2.

The tree structure offered a solution to the technical problem of representing "hi-

[123]An original term Newell and Simon used to talk about the search for proofs in the context of the Logic Theory Machine was *maze,* in analogy with behavioral psychological experiments with rats in mazes. Simon drew on such experiments extensively, and used the metaphor of a maze in many of his publications about decision making processes in individuals and organizations. For a discussion of Simon's use of "maze" metaphors, see Crowther-Heyck, *Herbert A. Simon*, esp. Chapter 5 "*Homo Administrativus*, or Choice Under Control", pp. 98 - 119. The observation that the word "maze" preceded the use of "tree" in their reasoning about proof search comes from Roberto Cordeschi, *Discovery of the Artificial*, p. 179. It is difficult to discern for sure from the archival materials how the use of the word "tree" replaced or accompanied that of "maze" with any certainty. However, by 1958 in "The Processes of Creative Thinking" (Presented at a Symposium on Creative Thinking, University of Colorado, Boulder, Colorado, May 16, 1958., also RAND Technical Report No. P-1320), they retain the use of the word "maze" to denote a problem search space as a whole in later works, but the tree appears as the dominant visualization of computational behavior.

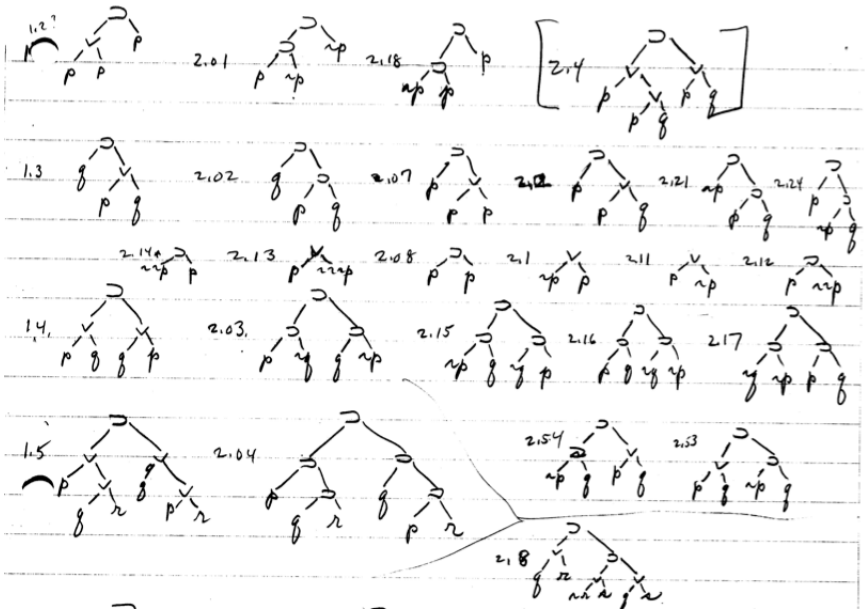


Figure 1.5: Hand-drawn Logical Expression Trees by Simon. CMU-HS, RAND Series, "Graphic Materials", "Logic Theorist", not dated.

erarchy" of operations in a logical proposition without the need for a system of parentheses. In Figure 1.4, for example, the operator "implies" is the main operator for the proposition, whereas "or" is a nested operation. In human-oriented symbol systems, the hierarchy of operators is typically an implicit property of ordering and parenthesis or periods. For computers, however, explicit properties are preferable. In a tree, that hierarchy is built in to the structure explicitly, requiring no parsing operations thus minimizing the steps a computer would have to take to extract information about a given proposition. However, the imagining of logical expressions *as trees* may have also been facilitated by the fact that Newell and Simon were already thinking about the Logic Theory Machine as following a tree-like search path according to its heuristic methods. In some of Simon's hand-simulations of the Logic Theory Machine we can see both representational mechanisms at work at once. In the simulated proof of *Principia* 2.0 shown in Figure 1.6 for example, mini-trees consisting of the main operators of the

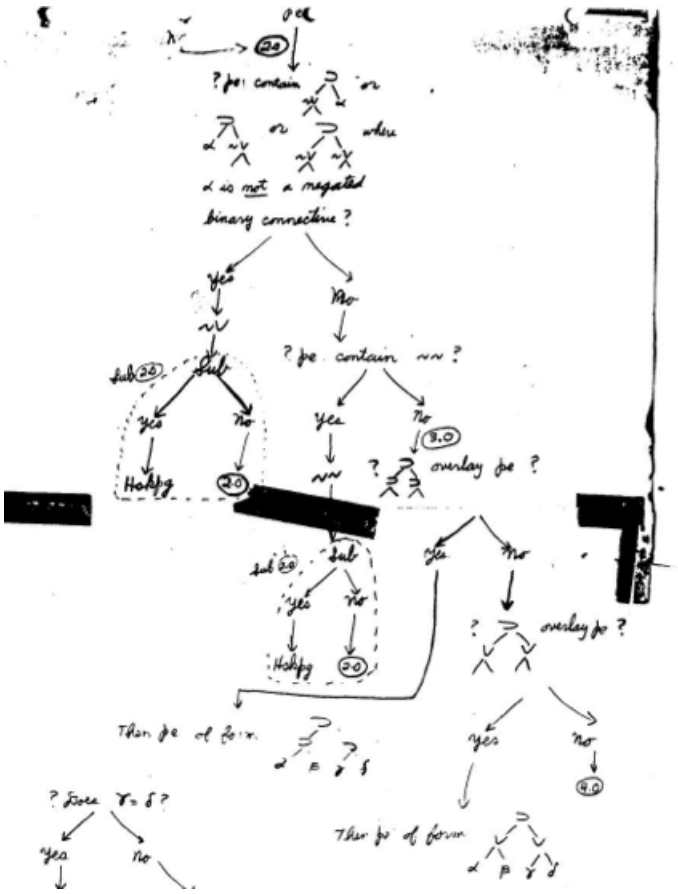


Figure 1.6: Hand-drawn Proof Trees by Simon. CMU-HS, RAND Series, "Graphic Material", "Logic Theory Machine."

relevant propositions are themselves laid out in a tree structure to show the generation of subproblems.

Polya's heuristics supplied Newell and Simon with a *model* of human theorem-proving practice - proof was a heuristic search by way of backwards, branching, subproblem chaining.[124] But models don't compute. This model still needed to *imple-*

[124]The identification and implementation of heuristics in the automation of reasoning tasks was taken up widely within the newly emerging Artificial Intelligence community, in no small part because Newell and Simon presented their work on the Logic Theory Machine at the now infamous 1956 summer conference at Dartmouth University. The phrase "Artificial Intelligence" itself was first coined by John McCarthy in conjunction with the planning of that conference. The Logic Theory Machine was the only running program presented at that conference and its basic approach was widely emulated by other early AI researchers. However, in spite of the clear and explicit influence of Polya, his notion of a heuristic and his belief that heuristics can be made formally explicit, his influence on AI after

mented. Implementation stands at the interface between models and machines - between abstraction and materiality. Newell, Shaw, and Simon described it this way:

> The Logic Theory Machine, of course, is a program, written for the JOHNNIAC computer, represented by marks on paper or holes in cards. However, we can think of Logic Theory Machine as an actual physical machine and the operation of the program as the behavior of the machine. One can identify Logic Theory Machine with JOHNNIAC after the latter has been loaded with the basic program, but before the input of data.[125]

That is to say, the Logic Theory Machine lived on paper and on punched cards. But it could be imagined as a machine in it own right - the JOHNNIAC was *transformed into* that machine by the input of the program instructions to its memory.[126] Implementa-

the Logic Theory Machine was somewhat tenuous. Of the many heuristics that Polya details in his many volumes, I have not found a single one besides "working backwards" and "subproblem chaining" that was actually implemented in any mathematical program. In a talk at a conference on heuristics in 1980, Newell explicitly addressed and lamented the generally indirect relationship of AI to Polya's work. Newell and Simon also primarily used a different source of heuristics than Polya's volumes in their own work - introspection on themselves, and the study of humans "reasoning aloud" in their enactment of cognitive tasks. Simon reports that even prior to their choice of *Principia* he had been "doing a lot of introspecting about [his] own problem-solving processes" (see McCorduck, *Machines Who Think,* p. 162. Roberto Cordeschi reports on their later adopted research model in which humans would "reason aloud" during problem-solving, offering a kind of introspective access to their ways of guessing, judging, and deciding within different problem domains. See Cordeschi, *The Discovery of the Artificial*, p. 181. In spite of many obvious problems, introspection is often used as a method for identifying human reasoning practices in automated theorem proving work. Other examples will be discussed in future chapters. It is worth noting that, to my mind, all introspection is a narrativization - a story telling based on the esoteric experience of one's self. For information on the planning of the Dartmouth Conference in Artificial Intelligence, see RAC-RF, Record Group 1.2, Series 200, Box 26, Box 531. For a discussion of this conference see Ronald Kline, "Cybernetics, Automata Studies, and the Dartmouth Conference on Artificial Intelligence" in *IEEE Annals of the History of Computing*, Vol. 33, No. 4 (2011): 5 - 16. See Pamela McCorduck, "The Dartmouth Conference" in *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence* (Natick, MA: A K Peters, 2004): 111 - 136 for a discussion of the conference based on oral histories revealing that many early AI practitioners perceived Newell, Shaw, and Simon as "ahead" by having a running program by 1956.

[125]Newell, Shaw, Simon, "Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics" [1957] reproduced in *Automation of Reasoning 1; Classical Papers on Computational Logic 1957 - 1966*, eds. J. Sikemann, G. Wrightson (Berlin: Springer-Verlag, 1983): 49 - 73 on p. 50.

[126]The JOHNNIAC computer was a stored-program computer designed in the image of the IAS machine, arguable the first to implement John von Neumann's architecture for storing both instructional data and input data in the same memory banks of the computer. The questions about the origins of stored program computing remain contentious. For example, I recently witnessed rather intense debates about whether the idea is already present in Alan Turing's 1936 paper "On Computable

tion, in this view, was the process of turning the JOHNNIAC into the Logic Theory Machine. Newell, Simon, and Shaw needed a way to represent logical propositions and inference rules inside the JOHNNIAC that would enable it to perform subproblem chaining with its bits of memory and basic computational capacities.

Two main features characterized their efforts. The elements of logic – its propositions, axioms, rules of inference, and so on – had to be given a form that could be input to the JOHNNIAC and stored in its memory systems. Second, the heuristic rules in their model of human theorem-proving behavior, had to be translated into computer operations that manipulated the contents of that memory.

In developing these representations and operations, Newell, Shaw, and Simon had to navigate the materiality of the JOHNNIAC computer. Here, the analogy between minds and computer programs began to fall apart. Here, it wasn't the constraints on human reasoning that mattered, but the limitations of computing machines. Here, the programmers had to accommodate the affordances of the computer and in so doing, abandon, to an extent, the commitment to simulating human practice. Here is where the model met the machine.

Numbers with an Application to the Entsheidungsproblem" or whether the idea was new with John von Neumann's "First Draft of a Report on the EDVAC" in 1945. A very exciting intervention in these discussions is "Reconsidering the Stored Program Concept" forthcoming in *IEEE Annals of the History of Computing,* by Thomas Haigh, Mark Priestly, and Crispin Rope who have been revisiting the history of the ENIAC computer. They trace the many meanings and dimensions of the concept of stored program computing, and correct many gross oversimplifications of the term and the surrounding debate. I am very grateful to Thomas Haigh for sharing this work with me, and in general for discussions of stored-program computing and the history of software and mathematical software in general. Another helpful resource for this interested in the evolution of storage and data manipulation in early mainframe computing is *The First Computers: History and Architectures*, eds. Raul Rojas, Ulf Hashagen (Cambridge, MA: MIT Press, 2000). The edited volume does not have a chapter devoted to the JOHNNIAC, however the chapter on the IAS Machine (of which the JOHNNIAC was a copy) describes its overall design: William Aspray, "The Institute for Advanced Study Computer: A Case Study in the Application of Concepts from the History of Technology": 179 - 193.

Reformalism: Linked Lists and Information Processing

It was not clear to Newell and Simon from the start what would be involved in actually programming the heuristic theorem prover that they imagined. They operated in the absence of pre-made languages, compilers, libraries of functions, and associated practices that have streamlined implementation to a large extent today. Starting in 1955, Newell's notes include a constant return to very basic questions about just how they should proceed and what was needed for the project. Newell and Simon's early work in 1955 and 1956 indicates that two issues quickly became a primary focus in the development of Logic Theory Machine: the need for a *language* in which problems could be formulated, solved, and the results expressed, and the need for *memories* (i.e., ways of storing data) that would make all of the necessary information available to the program throughout its work.

For example, in January of 1955, Newell wrote the following preliminary notes that introduce these issues:

> The purpose of a problem solving program is to 1. Find out and formulate what the problem is. 2. Determine what methods might solve the problem. 3. Organize the solution. Several problems here: 1. What is available to diagnose the problem? retrace program - *reading symbols*. Storage of special info. [...] How to *express* the result when you get it? [...] Need a set of *working memories* which provide appropriate content for talking about problem. [...] What must it be able to do? Accept a problem from outside. Here *a certain language* is necessary. A content language would be preferable here so we could "talk" to the machine.[127]

Even in this early exploration where it is not yet clear what the basic structure of the program is going to be, certain key preliminary ideas are visible. Newell identifies the

[127]CMU-AN, Series I: RAND, Notes, "Logic Theory Machine I", Early Notes dated January 10, 1955: all my emphasis. The details of their language will be discussed in what follows, but I want to note here that from what I understand "content language" for Newell is something akin in spirit to what is now called a "declarative language" (Prolog is an example). That is to say, he wants a formal language with which he can convey a problem to the computer - a way of describing the problem to it - without necessarily involving an algorithm for actually solving the problem at the same time.

program in a very basic way as a "symbol reader", for which the processing of symbols is a part of the problem-solving process. In fact, as I will discuss shortly, *information* as Newell and Simon mean it (e.g., in "Complex Information System") was constituted by *symbols*.[128]

As I suggested in the section "The Possibilities of Computing" in the introduction to the dissertation, the stipulation that the Logic Theory Machine be able to read symbolic information was itself something of a novelty in early computing research. The mainframe computers of the 1950s were all originally built to do *numerical* work - to compile ballistics tables, to solve complex differential equations, to perform numerical simulations of nuclear chain reactions, and so on. Newell and Simon were among the earliest computer practitioners to imagine the computer as a more abstract agent, capable of manipulating any formal symbolic system whatever, whether its constituent symbols referred to a numerical domain or not.

However, the JOHNNIAC was built with numerical calculations in mind and the machine language with which it was usually programmed in the 1950s was designed with numerical calculations in mind. Programming the JOHNNIAC to be a symbol reader in a language whose commands and infrastructure focused on numerical calculations would have been extremely difficult if it was even possible. Newell, Shaw, and Simon wanted to turn the JOHNNIAC into the Logic Theory Machine, and to do so they would have to develop a new set of tools. To transform JOHNNIAC from a numerical machine to a symbolic machine, they elected to design a new programming language for communicating with the computer, fashioning the computer's behavior, and for

[128] Newell and Simon indicate a particular debt to some early work on the automation of chess playing by Oliver Selfridge and G.P. Dineen. Selfridge visited RAND in 1954 to demonstrate their work on chess playing and Newell has described their meeting as something of a revelatory experience (see Pamela McCorduck, interview with Allen Newell (1976), CMU-PM. In fact, Newell's first attempt to specify a symbolic program was in fact a chess player and not a theorem-prover

formulating problems, results, and practices in nonnumeric domains like logic.[129]

Newell, Shaw, and Simon developed a new programming language - the Information Processing Language (IPL) - for propositional logic within which the Logic Theory Machine would ultimately be realized. The first instantiation (originally called the "Logic Language", and later renamed IPL I) was worked out by Newell and Simon without concern for the technical specifications of the JOHNNIAC computer. IPL I was an example of a *reformalism* - in it, the axioms and permitted inferences of *Principia* were translated into what they called "Information Structures" and "Information Processes." IPL II was the first language designed by Newell and Shaw to actually program the JOHNNIAC computer to become the Logic Theory Machine. It was a transformation of the IPL I into commands and features that would allow the actual programming of the JOHNNIAC.[130]

The Logic Language and IPL II are the primary focus of this section, and they are the site in which Newell, Simon, and Shaw's reformalism of propositional logic, the creation of new ways of reading and writing logical expressions, and the changing materiality of mathematical objects can be most clearly seen. These languages were a condition of possibility and a constitutive technology for the Logic Theory Machine. With them, JOHNNIAC could be transformed into the Logic Theory Machine.

The first translation of *Principia* into an information processing system was executed by Newell and Simon, and presented in a relatively complete form for the first time in September of 1956.[131] Their first order of business was to articulate and for-

[129]A succinct reflection on these early tensions between the numerical intentions of early computing and the transition to symbolic tasks can be found in Herbert Gelernter, J. R. Hansen, C. L. Gerberich, "A FORTRAN-Compiled-List Processing Language" in *Journal of the ACM*, Vol. 7, No. 2 (1960). In the later 1950s, Gelernter lead a project at IBM research to program theorem proving in elementary plane geometry That project included an expansion and modification of the IPL language.

[130]The IPL language is best known for being the immediate predecessor to the LISP language - developed largely by John McCarthy and the emerging AI community in California later in the 1950s. LISP borrowed many features of IPL, including list processing which will be a focus in what follows.

[131]Newell, Simon, "The Logic Theory Machine: A Complex Information Processing System", RAND

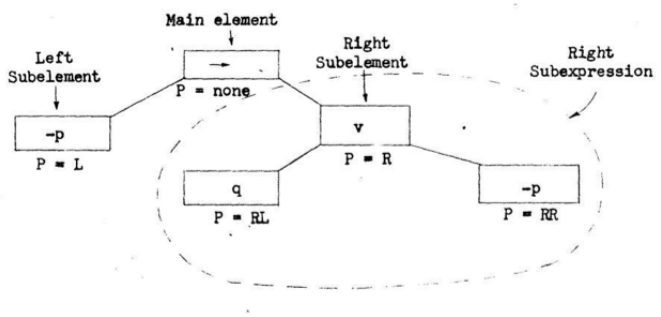


Figure 1.7: Tree Diagram of *Principia Mathematica* Proposition 1.7: $\sim p \rightarrow (qvee{-}p)$. Newell, Simon, "The Logic Theory Machine," p. 9.

malize the representation of expressions as "trees" in which logical operators - including "OR", "NOT", and "IMPLIES" - function as parent nodes for the variables that they relate as shown in Figure 1.7. However, in the Logic Language, each element in these expression trees would hold more information than their counterparts in *Principia*. Whereas the alphabet for Whitehead and Russell's logical expressions was limited to the symbols for variables, constants, and logical operators, Newell and Simon developed a set of eight additional symbols that would specify an element in the expression. Newell and Simon's symbols include the following:

- G: with value 1 it indicates that a variable is negated (like p is above) and with value 0 it indicates that a variable is not negated.
- V: indicates that an element in an expression-tree is a variable or not.
- F: indicates that an element is "free" for substitution - meaning, free to be replaced by definitionally or inferentially equivalent expressions.
- C: indicates that an element is a connective ($\bigvee$ or $\rightarrow$).
- N: is set to the *name* of every variable. Name here means the symbol that is used to represent that variable. In *Principia* 1.7, for example, q and p are the names of the two variables included.

Publication P-868 (July 12, 1956) [paper was presented at the *Symposium on Information Theory,* Sponsored by the Professional Group on Information Theory of the Institute of Radio Engineers, September 10, 1956].

- P: is a symbol that keeps track of the position of a given element in the tree. The parent node for any expression (i.e., the main operator) has no P value but every other element does. The P values are shown in the diagram above for the expression-tree of *Principia* 1.7. "P = RR" means an expression us the right child of a sub-tree that is the right child of a parent logical connective, for example.

The last two symbols for specifying elements in an expression tree are most important for my purposes here because both include information about *where* the expression is (rather than what it is):

- A: stores the *location* of the whole expression-tree in *memory*. In "The Logic Theory Machine", where Newell and Simon introduce these symbols, they are not yet talking explicitly about the JOHNNIAC memory.
- U: specifies whether or not an element is a "unit". By "unit", Newell and Simon mean that an element is held together in memory as a whole. More on this in a moment.[132]

In addition to these symbols, Newel and Simon also created three symbols for describing a tree as whole - H, J, and K - that specified the total number of variables in a tree, the number of distinct variables in a tree, and the number of "levels" (i.e., logical operators) in the tree.[133]

Those who work with traditional logical symbol systems might say that the information specified by these symbols is either *implicit* or *visually obvious* in notational systems like Whitehead and Russell's. And this may be true for trained human practitioners. However, the JOHNNIAC was not equipped with anything like a visual faculty and it, like all computing machinery, required direct and explicit access to all necessary information for solving a problem.

JOHNNIAC could not *see* that there are three variables, two of which are negated in *Principia* 1.7. I propose that this transformation of information in logical expressions

[132]Newell and Simon's articulation and definition of these symbols can be found in "The Logic Theory Machine", pp. 10 - 11.

[133]See Newell, Simon, "The Logic Theory Machine," p. 11.

from visually perceptible and implicit for human practitioners to explicitly represented by a symbolic alphabet is nontrivial. First, Newell, Shaw, and Simon understood reasoning (for humans and for computers) as *symbol manipulation* - as such, the addition of new symbols to the formalism of logical work literally changes the stuff of reasoning. It adds pieces to the information system that is understood to be logical thought. It also diverges from the symbolic notation system in *Principia*, meant to enable a particular kind of reasoning.

More than that, the addition of this new information - these new symbols - begged an organizational question: how to keep these expressions-as-trees with all of the symbols that accompany each element organized such that the computer can keep track of them, manipulate them, and ultimately prove theorems with them? One of the primary difficulties of implementation in the 1950s was the extremely limited memory then available. Across its various storage devices, the JOHNNIAC didn't even have *one tenth* of the memory of a floppy disk from the 1980s.

The JOHNNIAC had two different kinds of storage, both extremely limited in capacity according to today's standards: the faster memory was the magnetic core memory. Here, JOHNNIAC would store propositions it was actually working on - generating subproblems from, and so on. The slower and more abundant memory was magnetic drum memory where it would store other relevant information like previously proven theorems.[134] When the JOHNNIAC first became operational early in 1953, it had RCA selectron tubes for memory storage. However, these were replaced with magnetic core storage commissioned from the International Telemeter Corporation later in 1953 that was upgraded in 1955. Also in 1955, around the time that Newell

[134]This dichotomy of storage remains in modern computing technologies (in the form of RAM and hard drives respectively). This dichotomy was also reflected even in the high level Logic Language (IPL I). Newell and Simon specified that some memory structures would be "working memories" - in the magnetic core - and "storage memories" - in the magnetic drums. See Newell, Simon "The Logic Theory Machine", p. 11.

and Simon began work on the Logic Theory Machine, a magnetic drum storage unit was added to the JOHNNIAC for additional storage.[135] The magnetic core offered 4096 words of memory (each consisting of 40 bits), and the magnetic drums provided 9 216 words.[136] In total, the JOHNNIAC included 532 280 bits (roughly equivalent to 65 kilobytes by modern measure) in which all Newell, Shaw, and Simon's reformalized propositional calculus - all of its protocols, instructions, objects, and processes - would take up residence.[137]

Given the extremely limited storage of 1950s computers, practitioners were required to address many difficult "memory management" problems in the process of implementation. How to keep track of available memory? How to organize relevant data in memory? How to reclaim memory once it's no longer in use? Questions like these were (and in fact still are) central to the work of implementation. Newell, Shaw, and Simon needed a way to store and organize all of the relevant information for logic proof by backwards subproblem chaining in JOHNIAC's limited capacity. Their solution to this question is at the heart of what interests me in this chapter: in order to keep all of these new symbols organized and usable, they devised a new information structure called a *linked list*.

Each of the eleven symbols that specified an element in a logical expression (a

[135]These details are provided in Willis Ware, "Johnniac Eulogy", RAND document P-3313 (March 1966), pp. 7 - 8.

[136]In his "Johnniac Eulogy," Ware indicated that the magnetic drum could store 12 000 words. However, Newell and Shaw reported that 9 216 words were available to Logic Theory Machine on the magnetic drums. See Ware, p. 8 and Newell, Shaw "Programming the Logic Theory Machine", p. 7.

[137]Magnetic core memory consisted of small rings of magnetic material woven together by electrical wires. Electrical current running through the wires served as a mechanism for setting the direction of the magnetic field produced in those rings in one of two possible directions. Those two directions of magnetic field served as the 1s and 0s of the binary logic that constitutes all digital computation. A more in depth exploration of the history and functioning of magnetic core memory is available in Paul Ceruzzi, *A History of Modern Computing* (Cambridge, MA: MIT Press, 1991 [2003]), esp. pp. 49 - 53. Magnetic drum memory operated according to similar mechanisms, except that drums stored data on the magnetic drum-surface using read-write heads that would direct discrete magnetic fields. A more in depth exploration of the history and functioning of magnetic drum memory is available in Ceruzzi, *A History of Modern Computing*, esp. pp. 38 - 44.

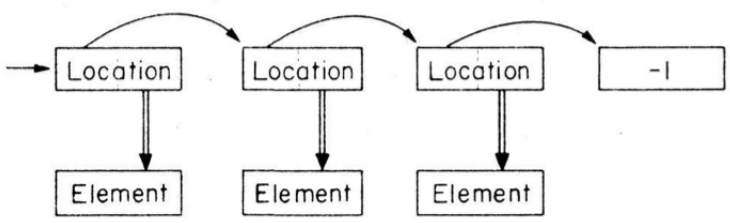


Figure 1.8: Diagram of a Linked List Information Structure. Newell, Shaw, "Programming the Logic Theory Machine", Rand Document P-954 (February 1957), p. 13a.

variable or a logical operator) would be stored in a contiguous set of bits of Johnniac's memory. The last of those bits would be used to store the numerical *address* of the next element in the list that could be stored *anywhere else* in Johnniac's memory. Here, the elements of a logical proposition were not concatenated in sequence together but were rather distributed throughout JOHNNIAC memory, held together by a virtual chain of address pointers. In this form, they could not be registered as a whole by sight or any other mechanism, but only by the *process* of traversing the pointers. In their article devoted to the actual implementation of the Logic Theory Machine on the JOHNNIAC, Newell and Shaw diagrammatically represent the list structure as seen in Figure 1.8.

Linked lists solved one particular memory management problem. They enabled the JOHNNIAC to make use of whatever memory happened to be available when transforming, manipulating, or storing logical expressions. Rather than finding chunks of contiguous memory in which to write a whole logical expression, or worse, to create that contiguous memory, the JOHNNIAC could store new information anywhere that there was space available and simply point to them with the appropriate address.

Imagine, for example, that you want to keep an updated guest list for a party. Instead of finding a new sheet of paper and writing the whole list out anew with one fewer name each time someone replies in the negative, it would be far less resource-

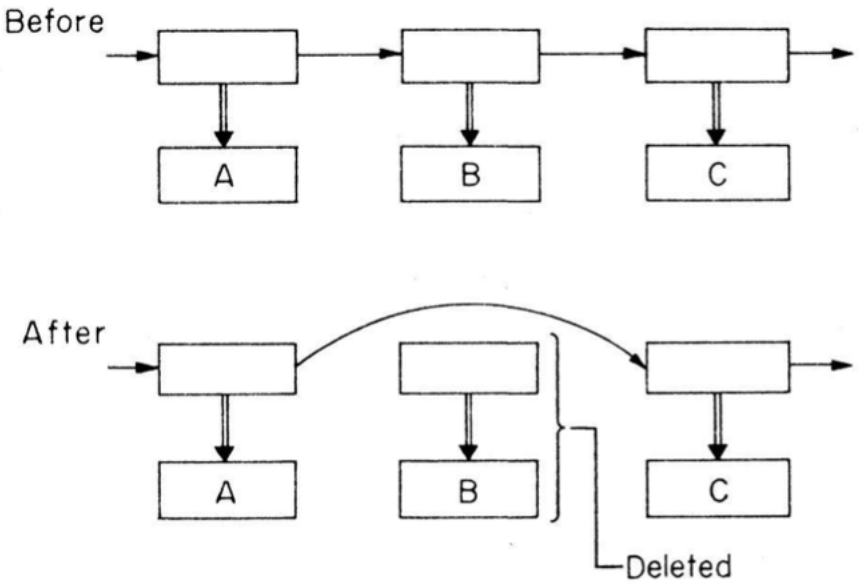


Figure 1.9: Diagram of Deleting an Element from a Linked List. Newell, Shaw, "Programming the Logic Theory Machine," p. 14a.

consuming to simply cross a name off the list. So it was with linked lists, except that they were not lined up like on a page: if the JOHNNIAC wanted to remove an element from a logical proposition, it simply rerouted the address pointer from the previous element to the following element, as shown in Figure 1.9 rather than rewriting the whole expression without it, after finding available memory to do so. In this transition, the element B was removed from a logical expression and all that was required was for the element A to edit its address indicator to point to C instead. After that, the address of B would be added to a list of available memory that JOHNNIAC kept track of during any run of the Logic Theory Machine program. Only two readdressing operations were required rather than all of the housekeeping computational work of finding or creating enough memory to rewrite the transformed expression as a whole.

The list structure helped both with the reuse of freed memory and with the project of keeping memory structures in tact over transformation because the elements in a

given list need not be *spatially* together in memory. That is, the actual magnetic bits within which the elements of a list are stored do not need to be neighbors in the magnetic core or drums. This is because every unit in a list keeps track of its successor's address no matter where it might be.[138]

This feature of the list structure - that lists need not be physically located together - points to just how different this formalism for logical propositions is from the notational system developed in Whitehead and Russell's *Principia.* Their symbolic notation represented logical expressions as concatenated inscriptions meant to reveal patterns and enable high-level abstraction in the human mind. For them, spatial "togetherness" was a central feature of their notational system for propositional logic: "The terseness of the symbolism enables a whole proposition to be represented to the eyesight as one whole, or at most in two or three parts divided where the natural breaks, represented in the symbolism, occur. This is a humble property, but is in fact very important in connection with" developing intuition and cognition of abstract ideas.[139]

Human practitioners relying on tractable spatiality of logical expressions would be hard pressed to make the dispersed list structure work for them in the search for a proof. In fact, Simon - whose many hand-simulations of the Logic Theory Machine survive in his archive collection - never once simulated the program using linked lists but always used the tree formalism. Linked lists were computational objects. They were dynamic and could only be taken in as single objects by way of a *process* - traversing the address pointers one after another. They accommodated the computer, not the person.

Human engagement with the list information structure is at the level of diagrammatic representation and protocol design not the level of use. Newell and Shaw devel-

[138]The only requirement was that an expression be held entirely in the "working memory" (i.e. the magnetic core) or in the "store memory" (i.e. the magnetic drums). Lists cannot span those two memory media.

[139]Whitehead, Russell, *Principia Mathematica*, Vol. 1, on p. 3.

oped the instructions by which JOHNNIAC would construct and use lists, but would not be able to make more than trivial use of them themselves. In this regard, they are a technology for understanding and working with computers rather than for understanding and working with logical expressions.

The linked list structure should be taken seriously and literally as a new representation system for propositional logic. Linked lists in magnetic core and magnetic drum memory were a new mode of material being for the propositions of elementary logic. And, following the contributions of media scholars like Matthew Kirschenbaum who propose that digital writing should be understood as *literal writing* (albeit post alphabetic), we could go so far as to say that the list represented a new way of *writing* mathematics.[140] Linked lists were a new formal and material tool devised at the intersection of incumbent ideas about theorem-proving and the material reality - the technical capacity - of the JOHNNIAC computer.

If these list structures are understood as a new digital way of *writing* logical expression in the memory storage of the JOHNNIAC, how are these inscriptions *read*? What does the JOHNNIAC-as-Logic Theory Machine *do* with this reformalism? Newell, Simon, and Shaw also needed to transform their heuristic rules and the inference rules from *Principia* into protocols for the algorithmic manipulation of linked lists. Herein lay another reformalism.

In Newell, Simon, and Shaw's reading of *Principia* there are two primitive inference rules that are needed for the construction of proofs in propositional logic: the "rule of substitution" and the "rule of detachment".[141] Additionally, Newell, Simon, and Shaw

[140]Kirschenbaum, *Mechanisms*.

[141]See Newell, Simon, "The Logic Theory Machine", pp. 25 - 26. See O'Leary, "The propositional logic of *Principia Mathematica* and some of its Forerunners" in *Russell: The Journal of Bertrand Russell Studies*, Vol. 8, No. 1, (1988): Available at: http://digitalcommons.mcmaster.ca/russelljournal/vol8/iss1/10, on p. 94. Newell, Simon, and Shaw's reading of *Principia* is expressly informed by the introduction to propositional logic written by Hilbert and Ackerman, "Chapter One" in *Principles of Mathematical Logic* (New York: Chelsea, 1950) in which

had created a model of heuristic search by way of working backwards and subproblem chaining. These rules, the former taken to be primitive processes of propositional logic, and the latter, taken to be basic elements of human practice, had to be translated into the language of linked lists. To this end, Newell, Simon, and Shaw crafted forty-four primitive "information processes" to specify the full behavior of the Logic Theory Machine, including the two inference rules. These forty-four rules were designed with linked list processing in mind and were organized into the following eight types:

- "Find" instructions, that locate addresses in memory;
- "Store" instructions, that move data from the working memory (magnetic core) to the storage memory (magnetic drums);
- "Put" instructions, that specify the movement of symbols within the working memory;
- "Numerical" instruction, that execute arithmetic operations, e.g. on the numerical data involved in the specification of a tree;
- "Assign" instructions, that govern the writing of new symbols in working memory;
- "Compare" instructions that, that enable the Logic Theory Machine to check if different symbols or values are equal;
- "Test" instructions, that enable the Logic Theory Machine to check different properties (i.e. values) of list symbols;
- "Brach" instructions, that enable the Logic Theory Machine to enlarge lists and point to new memories.[142]

The forty-four variations of these eight types of instruction constitute the information processing that the Logic Theory Machine was capable of performing. With them, it would manage the construction, manipulation, and deletion of lists in working and

those distinctions are made explicit (see pp. 28 - 31). They were, in turn, working with an earlier distillation of the rules of inference offered by David Hilbert and Wilhelm Ackermann in 1928.

[142]Newell and Simon detail these types of instructions in "The Logic Theory Machine", pp. 17 - 24.

storage memory. Here was a "reformalism" in which the rules of inference and heuristic rules were translated into the formal and material character of computing.

Of course, Newell, Shaw, and Simon recognized that to some extent, JOHNNIAC was different from a person, conceding that "there are many details of LT that we would not expect to correspond to human behavior. For example, no particular care was exercised in choosing the primitive information processes to correspond, point by point, with elementary human processes."[143] Instead, the "primitive information process" were chosen to accommodate the computer.

And yet, the Information Processing Language, and with it, linked lists and list processing, became the main tool kit for their continued work on Artificial Intelligence. It came to epitomize the "information processing model" that they took to characterize cognition, reasoning, and (bounded) rationality. Moreover, the low level differences didn't prevent the collaborators from claiming that their programs were simulations of human behavior. The IPL, developed in response to the material constraints of the JOHNNIAC, became a central frame within which Newell, Shaw, and Simon understood AI and with it, human reasoning. It was not epistemologically neutral. It was not merely a representation. It shaped how they thought about minds and computers, it characterized their interactions with computers, and it gave rise to the development of many other tools that would be used for AI, including its more famous list-processing descendent, the LISP processing language developed during the 1960s at MIT and put to work in countless AI programs during the second half of the twentieth century.

[143]Newell, Shaw, Simon, "Elements of a Theory of Human Problem-Solving" in Psychological Review, Vol. 65, No. 3 (1958): 151 – 166, p. 154.

What's in a Linked List?

IPL and its constituent elements weren't just epistemologically significant for the study of minds-as-information-processors. They mattered for logic as well. Linked lists lived in JOHNNIAC's memory, yes. But they *also* lived on paper where they were invented, devised, explained, and explored by the human practitioners who imagined them into being. Indeed, it was in the archives, on paper, where I discovered them as well - it being impossible for us to ever see a linked list in computer memory at all. The architects of the Logic Theory Machine surrounded themselves with paper in new ways, they used paper to represent new things to themselves in new ways.

By asking after the character of linked lists, by looking for their origins, and by following them to their different sites and instantiations, we saw how complicated it was to put even a tiny part of the world into a computer. We uncover some of the difficult involved in rethinking the world through a materiality largely inaccessible to human experience and sensibility. And we saw new ways that the work of theorem-proving was done in the early years of this digital age.

At the same time as the program was being worked out and implemented, new tools for *people* were also being developed. Computers never replace human labor, they always transform and displace it. In one sense, the linked list replaced paper. Logical propositions were no longer inscribed sequentially on the page to be taken in and compared synthetically by human vision. They were inscribed in JOHNNIAC memory, dispersed, scattered, held together by a chain of address pointers. There they could not be seen by a human practitioner and there they could not be taken in as a whole at a glance, but had to be traversed according to list processing operations into which the logical rules of inference and demonstration were transformed.

However, they were invented and implemented and explained and described and

constructed by way of *diagramming*. The figures from the earlier sections of this chapter are pictures of *paper* - the ways of drawing and writing that accompanied the development and implementation of linked list structures. Newell, Simon, and Shaw did not get rid of paper. They put paper to work in a new way.[144] They developed linked lists diagrammatically on the page.

I claim that linked list diagrams are very interesting relative to the question of representation. What do linked list diagrams represent exactly? They represent two things *at the same time*: they represent both logical propositions *and* JOHNNIAC memory. They represent the thing to be digitized and the digital media together. This hybrid or dual representational scheme points to part of what is involved in carving out a place for computers in knowledge-production. Practitioners had to find ways of thinking about what the computer is, of representing it, and of working with it *at the same time* as they re-imagined their part of the world, their objects of interest, as digital things. Linked list diagrams are a trace of how mathematical objects were reconceptualized through the lens of computing at the same time as computing machines were being re-tooled and outfitted for mathematics. Linked lists were not merely representations of logical propositions, they were representations that introduced discrete, digital, computational, algorithmic, and processual properties to them.[145] Linked lists hybridized properties of logical propositions and Johnniac memory.[146]

[144]This is in fact a common phenomenon in the history of computing. Recent media scholars have shown how the "paperless office" is in fact a myth and that paperless offices are populated with all kinds of paper and paperwork. See for example, Abigail J. Sellen and Richard H.R. Harper, *The Myth of the Paperless Office* (The MIT Press, 2003); Kirschenbaum, Matthew G. "Editing the Interface: Textual Studies and First Generation Electronic Objects" in *Text: An Interdisciplinary Annual of Textual Studies* 14 (2002): 15-51.; Mark Priestly and Thomas Haigh are also currently working on the forms of flow diagramming that accompanied early uses of the ENIAC computer - often associated with the "programming by plugging" practices that predated punch tape and punch card programming.

[145]For a more philosophical exploration of the "dynamism" of computational objects, see Brian Cantwell Smith, *On the Origin of Objects* (Cambridge, MA: The MIT Press, 1996).

[146]A related claim is made by David Nofre, Mark Priestly, and Gerard Alberts in "When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950 - 1960" in *Technology and Culture,* Vol. 55, No. 1 (January 2014): 40 - 75. They explore how early

Some fascinating scholarship has been produced by anthropologists and historians who are interested in the role that written symbols and notational systems play in the history of human cognition. One early study was anthropologist Jack Goody's work in *The Domestication of the Savage Mind*.[147] He argued that the minds of traditional peoples and the minds of Western Europeans were not separated by some essential difference as had been proposed by his more colonially minded predecessors and colleagues. Instead, he believed that the cognitive abilities of all peoples were a product of their "technologies of literacy." He argued that ways of writing, recording, manipulating, and circulating information enabled and constrained memory, reason, and cognition. The "list" is one such literary technology that he identifies as important in the history of commerce, logical reasoning, and experiences of temporality. In a section called "What's in a list?" he writes:

> The list relies on discontinuity rather than continuity; it depends on physical placement, on location; it can be read in different directions, both sideways and downwards, up and down, as well as left and right; it has a clear-cut beginning and a precise end, that is, a boundary, an edge, like a piece of cloth.[148]

In a different vein, Hans-Jörg Rheinberger has emphasized the epistemological significance of note-taking and data recording practices among experimental scientists in determining the outcome and understanding of what was shown in an experiment.[149] Some scholars in philosophy have suggested that, in fact, if certain symbol systems are so inextricable a part of cognition, perhaps they should not be understood as *aids* to

programming languages had a kind of dual representational function in that they had to represent to the computer and they had to represent to the programmer. Early languages combined the needs of programmers and machines.

[147]Jack Goody, *The Domestication of the Savage Mind* (Cambridge University Press, 1977).

[148]Goody, *The Domestication of the Savage Mind*, p. 81.

[149]Hans-Jörg Rheinberger, "Scrips and Scribbles" in *MLN* Vol. 118, No. 3 (2003): 622 - 636.

cognition but rather as *part* of cognition itself.[150] Lists and other literary technologies like tables, graphs, and diagrams have received interest more recently as well among historians of science, technology, and economics. These accounts are often motivated by an interest in the *kind of thinking* that particular inscriptions enable or exclude. Data structures like linked lists can be thought of as new literary technologies that enable new ways of "thinking with computing."

The drawing and diagramming techniques deployed by Newell, Simon, and Shaw to represent logical expressions don't just point to new materiality for logical expressions - they point to new *thinking* about logical expressions. And in particular, Newell, Simon, and Shaw were not thinking about logical expressions in order to prove theorems directly, they were thinking about logical expressions in order to program a computer to prove theorems. The interventions of the technology in the cognitive goals and resources of its users are reflected in the new tools they design for working with old objects. Processes of automation seldom, if ever, *replace* human thought. Instead, automation attempts rather displace and transform human thinking at the same time as they enable the construction of new objects of thought - these develop always in tandem. And, as I will reiterate throughout in the dissertation I am less interested in the question of whether machines are thinking than I am in the changes in human thinking that surrounded the use of computing machinery.

Conclusion

Whereas I have elected to focus on the *differences* between Whitehead and Russell's notational system and inference rules and those developed for the Logic Theory Ma-

[150]This suggested, called the "extended cognition hypothesis" originated with Andy Clark and David Chalmers in "The Extended Mind" in *Analysis*, Vol. 58, No. 1 (1998): 7 - 19. Many discussions of and responses to the hypothesis are collected in Richard Menary, ed. *The Extended Mind* (Cambridge, MA: MIT Press, 2010).

chine, Newell and Simon rather emphasized profound similarity. According to their vision of reasoning-as-information processing, the human mind and the JOHNNIAC-as-Logic Theory Machine were engaged in the same kind of deductive reasoning. And, very interestingly, when the Logic Theory Machine's proofs were translated *back* into the human-friendly notation of Whitehead and Russell (e.g. every Logic Theory Machine execution of the list-processing routine for the three primitive rules), most of the Logic Theory Machine's proofs were the same as those constructed by Whitehead and Russell.[151] The program proved about fifty of the theorems in the early chapters of *Principia*. It was most celebrated for producing a previously unknown proof that Newell and Simon thought was "more elegant" than that presented by Whitehead and Russell. And Russell, by then Earl Russell, agreed.[152]

However, I have argued that in spite of their desire to preserve the form of proof presented in *Principia*, the development and implementation of the Logic Theory Machine in fact involved epistemologically significant transformations in the tools and practices for studying mind and for cognizing and working with logic. Newell, Shaw, and Simon crafted a new representational system and a new set of operations for proving theorems in propositional logic, and with them, they endowed that logic with new properties and behaviors. Heuristic search across linked lists according to forty-four primitive information processes was a different form of proof than the written, vision-oriented, paper-based ones of the early twentieth century.

[151]Newell's identification of this similarity is also found in his correspondence with Russell. See The correspondence between Simon and Russell is reproduced in Simon, *Models of My Life*, pp. 207 - 209.

[152]Simon documents their correspondence in his autobiography. Simon, *Models of My Life* (New York, NY: Basic Books, 1991): 207 - 209.

Chapter 2

"Mathematical Objects in Action": Implementing Herbrand's Theorem[153]

Introduction: A Different way to Prove Principia

"There is no need to kill a chicken with a butcher's knife. Yet the net impression is that Newell-Shaw-Simon failed even to kill the chicken with their butcher's knife."[154] This criticism was directed at the Logic Theory Machine, a mid-1950s program intended to prove logical theorems from *Principia Mathematica*.[155] The remark came from Hao Wang, himself the architect of a set of late 1950s programs for proving theorems from

[153]The phrase "Mathematical Objects in Action" is from Edward Ng, "Introduction" in *Proceedings of the Symposium on Symbolic and Algebraic Computation* (Berlin: Springer, 1979): p. 2. In describing the subject matter of the *Symposium on Symbolic and Algebraic Computation*, Ng wrote that "There is presented in this symposium a rich variety of mathematical concepts from number theory, algebraic geometry, differential algebra, group and field theories. There is a diversity of mathematical objects in action." The Symposium was part of a particular approach to computing - one that sought to harness the computer for nonnumeric and algebraic work rather than numerical computation.

[154]Hao Wang, "Toward Mechanical Mathematics" in *IBM Research Journal*, Vol. 4, No. 1 (January 1960): 2 - 22, 4. The paper was received at the end of 1958.

[155]The Logic Theory Machine is the subject of Chapter One: Rewriting *Principia*: Implementing Intelligence.

Principia.[156] His criticism was born, in part, from the numbers. The Logic Theory Machine successfully proved thirty-eight of the first fifty-two theorems of *Principia*, failing in fourteen cases.[157] Wang's programs did much better. His first and least complex program alone, called the "System P," proved every one of the more than two-hundred and twenty theorems in the first five chapters of *Principia* and in considerably less time.[158] His later program, called the "Program P" "disposed" of the over three

[156]Wang designed three programs all together, between 1958 and 1960. Each of the programs was designed to tackle a specific subset of the logic of *Principia*. The first program was for the propositional calculus - the domain in which the Logic Theory Machine also worked. The second program was designed to construct well-formed propositions in the propositional calculus and distinguish trivial from interesting ones, and the the third was a theorem-prover for all of the predicate calculus with equality that could prove upwards of 85% of all theorems in the whole of *Principia* in about an hour. I will discuss some of these distinctions and the strategies Wang deployed in automating theorem-proving therein in what follows. Wang developed these programs first while working at IBM research labs in the summer of 1958, and later at Bell Research Labs in 1959 and 1960. Wang, "Towards Mechanical Mathematics" in *IBM Journal of Research and Development*, Vol. 4, No. 1 (1960): 2 - 22; Wang, "Proving Theorems by Pattern Recognition Part I" in *Communications of the Association of Computing Machinery*, Vol. 3, No. 4 (1960): 220 - 234; Wang, "Proving Theorems by Pattern Recognition Part II" in *The Bell System Technical Journal*, Vol. 40, No. 1 (January 1960): 1 - 42.

[157]These theorems begin in Part 2 "Immediate Consequences of the Primitive Propositions" of "Section A: The Theory of Deduction" in Volume I of *Principia*, pp. 102 - 113. In most of the fourteen cases, the Logic Theory machine failed because it encountered limitations of computing resources. Newell, Simon, and Shaw were able to show that the program was incapable of proving one of those fourteen theorems, and suspected it was not capable of proving one other. That is to say - there were theorems in the list such that no possible sequence of Logic Theory Machine operations would produce a proof. These statistics and their associated running times were reported in Newell, Shaw, Simon, "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristics" in *Proceedings of the Western Computer Conference* (1957): 219 - 230, on p. 225 and they were reproduced by Wang in "Toward Mechanical Mathematics" on p. 3.

[158]It is somewhat difficult to compare computing times in the 1950s. Actual running times varied in large part because of significant differences between computing machinery. Much of the time required to run a program was tied up in processes of input and output and these depended on what mechanisms and media were used; the Johnniac mainframe on which the Logic Theory Machine ran was, for example, a much slower machine than the IBM 704 on which the System P ran. Comparisons of actual running time were not, therefore, measures by which to compare the *fundamental* efficacy of different programs. Many practitioners reporting their results at this time therefore distinguished between the "actual computing time" of a program's execution and the "total running time," the former excluding the time taken to input and output relevant data. Around this same time, computing practitioners were working out formal methods for measuring the running speed and complexity of algorithms in the abstract, independent particular implementation - here programs are represented as algorithms that run on an abstract mechanism like a Turing Machine, rather than an actual computing machine. This provided a standardized way for calculating the number of steps the algorithm would perform for an input of a given size, and comparing them across algorithms. Usually, this area of research is traced back to the work done by Michael O. Rabin at IBM Research Labs in 1958, first circulated as "Degree of Difficulty of Computing a Function and Hierarchy of Recursive Sets," *Technical*

hundred fifty theorems in the first nine chapters of *Principia* in less than ten minutes.[159] Wang's programs produced more proofs in less time.

Wang was a Chinese-American logician and philosopher, who came to the United States in 1946 to do a PhD under Willard Quine at Harvard University. He worked in mathematical logic studying the character of formal axiomatic and deductive systems. And it was to certain results of mathematical logic that he turned in designing his theorem-proving programs. Wang wanted to *implement* results from mathematical logic on a computer. His programs were based on Herbrand's Theorem. Herbrand's Theorem was a fundamental result of proof theory, a branch of mathematical logic aimed at proving theorems about proofs themselves by studying the formal properties of axiomatic, deductive and inferential systems.[160] The theorem, which I will discuss at length in "The Herbrand Universe" below, offers a procedure for proving certain theorems from one branch of logic - the predicate calculus - using another, simpler branch of

Report No. 2, O.N. R., (Hebrew University, Jerusalem: 1960). The study of algorithm running time, complexity, and efficiency has become a central tenet of theoretical computer science. The ability to perform generalizable speed comparisons is another reason why implementation has been relegated to secondary status in computer science, whereas algorithms are considered fundamental. Wang makes no mention of the complexities of comparing running times, and does not speak to differences between the Johnniac and the IBM 704. However, in spite of these complexities, there is no question that the Logic Theory Machine was much slower - an order of magnitude slower - at proving fewer theorems than the System P. In order to give a sense of the computing time involved in these results, Newell, Simon, and Shaw reported that the program took twelve minutes to provide a proof of theorem *2.45 and took twenty-three minutes to report a failure in attempting to prove *2.31 - these being representative of the times taken for similar cases. Conversely, the System P proved two-hundred plus theorems in twenty seven minutes total, three minutes of a "actual computing time." Only thirty seconds were required for the System P to discharge all fifty-two theorems on which the Logic Theory Machine was tried, three seconds for *2.45 and six seconds for *2.31. The comparative results were published by Wang himself in "Towards Mechanical Mathematics", p. 4 and also reported and analyzed by Donald Loveland in "Automated Theorem-Proving: A Quarter-Century Review" in *Automated Theorem Proving: After 25 Years*, [Contemporary Mathematics, Volume 29], ed. Woody Bledsoe, Donald Loveland (Providence, RI: American Mathematical Society: 1983): 1 - 46, pp. 6 - 7.

[159]Wang, "Proving Theorems by Pattern Recognition I" in *Communications of the ACM,* Vol. 3, No. 4 (1960): 220 - 234, p. 225.

[160]Proof Theory originated with Hilbert's program of axiomatization in the early twentieth century. For a historical survey of the emergence of proof theory, see Ivor Grattan-Guiness, *The Search for Mathematical Roots, 1870 - 1940* (Princeton, NJ: Princeton University Press, 2000) esp. 207 - 218.

logic - the propositional calculus.[161] However, when it was first demonstrated in 1930, the procedure could almost never actually be *used* for theorem-proving. The theorem served to establish a particular and surprising relationship between two branches of logic but it was not actually a tool for proving theorems on the ground. This kind of procedure was common in twentieth-century mathematical logic. For example, many logicians aimed to devise what were called "decision procedures" that, given some input statement, would "decide" if the statement was a theorem in a finite number of algorithmic steps.[162] These procedures offered insights into the formal structure of and the relationships between different domains of logic but few of them were actionable tools for actually proving that particular statements are theorems.

The basic idea of Herbrand's Theorem is this: for any given statement P in the predicate calculus, the theorem tells you how to construct a corresponding infinite series of statements in the propositional calculus, $S_1, S_2, S_3, \ldots$. It then tells you that P is a theorem if, and only if, for some number N, the first N statements in that series connected by the OR operator ($S_1 \vee S_2 \vee \cdots \vee S_N$) is a tautology, meaning that it is true no matter what values are assigned to the variables in $S_1, S_2, S_3, \ldots$.[163] So one way to search for a proof would be to test if $S_1 \vee S_2 \vee \cdots \vee S_N$ is a tautology for $N = 1, 2$, and so on. If P is a theorem, then eventually a tautology would be discovered (and therefore a proof), but this method is incredibly costly because N is usually some

[161] What makes Herbrand's Theorem powerful is that propositional logic is sufficiently simple as to be decidable, but predicate logic is not. That the decidable propositional logic can be used to prove certain statements in the predicate logic is therefore a significant and surprising result.

[162] One of Alan Turing's central results was to show that is is provably impossible to create decision procedures for sufficiently complex domains of logic. See Turing, "On Computable Numbers with an Application to the *Entsheidungsproblem.*"

[163] Each S_i contains some variables and for a particular S_i to be a tautology, it must be true under all possible assignments of true/false to those variables. The disjunction $S_1 \vee S_2 \vee \cdots \vee S_N$, however, is a tautology if, for under all assignments of true/false to all of the variables, *at least one* of the S_i is true. So, for example, it is not sufficient to simply check if each S_i is a tautology because it is possible for $S_1 \vee S_2 \vee \cdots \vee S_N$ to be a tautology when no particular S_i is independently. For example, given a single variable x, suppose that $S_1 = x$ and $S_2 = \sim x$ (where $\sim$ means "NOT"), then $S_1 \vee S_2$ is a tautology even though neither S_1 nor S_2 is.

astronomical number - for example, two trillion even for a simple example.[164] And worse, if P is *not* a theorem, then no tautology can ever be found and there is no way to know when to stop looking for one - you can't tell when to stop because you never know if checking just one more will make it a tautology. And although computers could certainly outpace their human counterparts in the execution of lengthy and complex operations, not even *they* could execute a method like this.[165] But Wang saw potential.

Wang wanted to *implement* Herbrand's Theorem on a computer - he wanted to transform the theorem from an abstract "in theory" kind of procedure to a tool kit computers could wield to actually prove theorems in predicate logic. He wanted to take insights offered by the theorem about logical structures and fashion them into computer operations. He wanted to make Herbrand's Theorem *useful* as more than a statement about the relationship between the two branches of logic. This would amount to an epistemic transformation of the theorem, changing what could be known with and through it. And he succeeded. The Program P is just such an implementation of Herbrand's Theorem, and it successfully proved all of the three-hundred and fifty plus theorems in the first nine chapters of *Principia Mathematica* in under ten minutes. Wang devised a method that he called "pattern recognition" with which the computer would exploit structural properties of $S_1, S_2, S_3, \cdots$ to decide if they do not contain a tautology (which would indicate that P was not a theorem). But these "patterns" were not ones that people could see and the method of recognizing them was not one that

[164]Wang offers one such example - a relatively simple problem that would involve checking $2^{48} - 1$ cases in "Proving Theorems by Pattern Recognition Part I," p. 222. For some subsets of the predicate calculus, it is possible to calculate an upper bound for N - a fact Wang makes use of in designing the Program P - but this was not included in Herbrand's original result.

[165]In 1954, three years before Wang made his programs, Martin Davis attempted to implement a decision procedure for additive number theory. However, Davis didn't modify the procedure to accommodate the affordances of the computer, and the program was only able to solve relatively simple problems as a result. Davis never published the program, but an account of it is provided in Davis, "A Computer Program for Presburger's Algorithm" in *Automation of Reasoning 1: Classical Papers on Computational Logic* 1957 - 1966, J. Siekmann, G. Wrightson eds. (Berlin: Springer Verlag, 1983): 41 - 48. Davis went on to become a central figure in automated theorem-proving research.

people could execute. These patterns lived in the digital memory of the IBM System 704 computer that Wang used at IBM to run his programs and they were processed by methods that were only possible on paper for the most trivial of examples. Wang implemented Herbrand's Theorem as a computer program constituted of operations that were beyond the reach of human execution.

Wang felt the superior performance of his programs sufficed as "conclusive refutation" of the Logic Theory Machine's design - his approach proved more theorems and faster.[166] His criticism would not have bothered it's creators Newell, Shaw, and Simon, however. The "proofs" that Wang's programs produced were *not the kind of proofs* that they were looking for. They wanted computer proofs that resembled human proofs. They wanted proofs that were constructed by steps of inferential reasoning inspired by human heuristics and intuition. They wanted computer proofs that captured and communicated human insight and practice. They wanted "proof in the meaning of Russell and Whitehead."[167] Indeed, although the Logic Theory Machine only proved a handful of theorems from *Principia*, perhaps its greatest success was that most of the proofs it output followed the same steps as those that Russell and Whitehead themselves constructed in the first decade of the twentieth century.[168] It produced proofs that, at a high enough level of abstraction, took the same inferential form as those that its human predecessors had drafted before it (although I argued in the previous chapter that they in fact introduced quite new forms of proof in spite of this).[169] Wang wanted

[166]See Wang, "Computer Theorem Proving and Artificial Intelligence" in *Automated Theorem Proving: After 25 Years*: 49 - 70, p. 53.

[167]Newell, Shaw, Simon, "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristic" in *1957 Western Computer Proceedings,* (1957): 218 - 230, p. 220.

[168]Herbert Simon was quick to identify this feature of the Logic Theory Machine's work when writing to Bertrand Russell to tell him of the program and its successes. Their correspondence is available in Simon, *Models of My Life*, pp. 207- 209.

[169]In the previous chapter, I argue that this high-level claim to "sameness" in fact hides significant differences at the level of implementation, practice, and materiality. However, when these other scales are ignored, the proofs produced by the Logic Theory Machine have the same form as those published in *Principia* in that they follow the same steps of inference rules. Once the "working backwards"

the computer to produce proofs that humans couldn't construct - maybe that humans couldn't even read. He wanted to usher in new forms of proof. In fact, Wang thought all of mathematics might be on the verge of transformation because of computers: he wrote that the "cross-fertilization of logic and computers ought to produce in the long run some fundamental change in the nature of all mathematical activity."[170] Newell-Shaw-Simon and Wang simply wanted different things from their computers and from proof.[171]

Why did Wang and Newell-Shaw-Simon disagree about proof in this way? Why did they want different kinds of proof from their computers? First, Wang and Newell-Shaw-Simon wanted different kinds of proof from computers because they thought very differently about what computers *are.* Through the lens of systems analysis (discussed in the section "Proof as Information Processing" in the preceding chapter), Newell and Simon perceived a formal similarity between human minds and computing machines. They wanted to instantiate existing human practices of theorem-proving on the computer as an example what they called "symbolic information processing" which they believed to be the defining activity of both human minds and modern computing machines.

Wang, on the other hand believed that human reasoning and computation were qualitatively different kinds of things. He agreed that some of what humans do would turn out to be automatable, but that "we are not likely to succeed in making the

process of the Logic Theory Machine has been successful, it can be presented in the reverse direction offering an inferential path from axioms to conclusion and many of these paths were the same ones presented by Russell and Whitehead. This was taken by Newell and Simon as evidence that the Logic Theory Machine was doing the same thing as Russell and Whitehead in proving theorems.

[170] Wang, "Proving Theorems by Pattern Recognition," pp. 232 - 233.

[171] Wang was quite dismissive of Newell and Simon's approach in his published writings. Newell and Simon were less so in press. However, John Alan Robinson (associated with Argonne and a subject of the next chapter, "A New Collaborator: Implementing Intuition and Inference") indicated that they were "quite rude" with respect to the logic approach to theorem-proving at conferences. Robinson, autobiographical talk, CADE 2012, Manchester England (July 2012).

machine imitate the man entirely."[172] More, *even if* computers could be made to imitate human mathematicians entirely, and Wang was skeptical that this would ever be possible, this would not be the best way to use them. Regardless of how much human practice they could be made to enact, computers could also do a great deal that people *can't* do and therein lay their true potential. They were quantitatively different as well. Computers were faster and more efficient at executing particular types of combinatorial, logical, and numerical tasks than their human counterparts. And Wang believed that *more is different* in this regard:

> The human inability to command precisely any great mass of details sets an intrinsic limitation on the kind of thing that is done in mathematics and the manner in which it is done. The superiority of machines in this respect indicates that machines, while following the broad outline of paths drawn up by man, might yield surprising new results by making many new turns which man is not accustomed to taking. [...] We are in fact faced with a challenge to devise methods of buying originality with plodding, now that we are in possession of slaves which are such persistent plodders.[173]

In Wang's telling, mathematics developed so as to accommodate the affordances of human reasoning which is limited, especially with regard to handling "great masses of details." He imagined that new ways doing mathematics would be possible if the different affordances of the computer were admitted and cultivated. Wang wanted to use computers to pursue proof practices that would be impossible for human mathematicians, but that would yield new insights and ideas about mathematics made visible only by the processing power of something like a computer. Newell and Simon were

[172]Wang, "Towards Mechanical Mathematics," p. 3.

[173]Wang, "Toward Mechanical Mathematics," p. 3. It bears noting as well that the provocative anthropomorphic language Wang and others used to describe the possibilities they perceived in computers open in to complex conversations. Here Wang calls computers "slaves" and elsewhere they are called "servants," "assistants," "colleagues," "mentors," and so on. These words invoke existing or historical experiences of social hierarchy and should not be read as neutral, even if intended as such. For a discussion of this language in technical fields, see Ron Eglash, "Broken Metaphor: The Master-Slave Analogy in Technical Literature" in *Technology and Culture*, Vol. 48, No. 2 (2007): 360 - 369.

interested in a sameness between minds and computers. Wang was interested in their differences.

But - if computers and people were in fact so different, how could people like Newell and Simon look upon the Johnniac mainframe computer and its cumbersome 1950s brethren and see something akin to the mind?[174] Wang thought that one might mistake the modern digital computer for a human mind because human minds were so often *idealized*, *abstracted*, and *theorized* as mechanical rule following entities. Wang wrote: "It seems as though logicians had worked with the fiction of man as a persistent and unimaginative beast who can follow rules blindly, and then the fiction found its incarnation in the machine."[175] Wang saw in the computer an actualization of a fictional and idealized model of people with all of their social, physical, and historical contingencies abstracted away. Computers weren't like people at all, people could never do what these machines did, but the machine was built in the image of an imaginary person.

This "imaginary mathematician" who doesn't sleep, eat, err, or die has made various appearances in the history and philosophy of mathematics, and Wang's reading of the computer as that fiction incarnate has repercussions for both. For example, philosopher

[174]I refrain from including John Clifford Shaw in this comparison. Although Shaw was integral to the implementation of the Logic Theory Machine, he did not himself subscribe to the belief that the Logic Theory Machine was a simulation of human reasoning processes. See Pamela McCorduck, J. C. Shaw Interview, June 16, 1975, (CMU-PM, Series III, Transcripts).

[175]Wang, "Computer Theorem Proving and Artificial Intelligence" in *Automated Theorem Proving: After 25 Years*, p. 67. The history of the "persistent and unimaginative beast" mythology of the mathematician is long and involved and it has many afterlives today. The long history of this vision would begin at least in the 17th century with the work of Gottfried Wilhelm Leibniz who sought a formal language capable of delimiting and describing all human thinking. See Matthew Jones, "Seeing All at Once" in *The Good Life in the Scientific Revolution: Descartes, Pascal, Leibniz and the Cultivation of Virtue* (Chicago, IL: Chicago University Press, 2006): 229 - 266. Later, George Boole described a binary logic that he believed captured the workings of human thinking in a rule bound formal system - indeed this logic was put to work in the actualization of digital computing machines and its original association with human thinking is often excluded from that history, See Boole, *An Investigation of the Laws of Thought On Which are Founded the Mathematical Theories of Logic and Probabilities* (1854). Different theorizations of mind as a machine are explored in P. Husbands, O. Holland, M. Wheeler, eds. *The Mechanical Mind in History* (Cambridge, MA: The MIT Press, 2008).

of mathematics Brian Rotman has proposed a model of "mathematical activity" based precisely on the distinction between ideal, symbolic, and procedural domains.[176] He suggests that the "figure of the mathematician" is in fact an "assemblage of agencies" including the actual human mathematician, the "Person," and the "Agent" who is imagined by the Person to execute procedures beyond his own capacity: "The Person makes a claim about an imagined task or procedure - counting, inverting a matrix, etc. - that the Agent will execute."[177] In Rotman's model, in order to formulate certain concepts or solve certain problems involving more steps than a person could execute, mathematicians imagine an "Agent" that could (e.g. count forever). More, for Rotman, the written symbol systems deployed in mathematical practice *make possible* the imagined Agent. It is by working with symbols like: "I II III IIII IIIII IIIIII ..." (including of course the ever-important ellipses) that such an Agent could be imagined.[178] He calls the Agent "a ghost, one that mathematicians invoke though they don't describe it so, when they *write* 1, 2, 3, ... and *think* infinity."[179] Herbrand's Theorem along with logical Decision Procedures, are good examples of how comfortable modern mathematicians are with the infinite and with procedures and processes that are not, in actuality, executable. They serve as good examples of Rotman's "Agent" at work. However, his model needs to be *historicized*.

Wang wanted to use the computer to *intervene* in the boundary between the "in principle" and the "in practice." He wanted to transform Herbrand's theorem into a set of computer-executable operations. These operations would not be executable by

[176]Brian Rotman, *Becoming Beside Ourselves: The Alphabet, Ghosts, and Distributed Human Being* (Duke University Press, 2008): esp. 59 - 62 and Rotman, *Mathematics As Sign: Writing, Imagining, Counting* (Stanford, CA: Stanford University, 2000), esp. 7 - 21.

[177]Rotman, *Becoming Beside Ourselves*, p. 61.

[178]Rotman proposes that other "imagined entities" - like ghosts and God - are also made possible by certain symbolic and semiotic systems. See, Rotman, "Ghost Effects" in *Becoming Beside Ourselves*, esp. 107 - 124.

[179]Rotman, *Becoming Beside Ourselves*, p. 130, my emphasis. Here in "The Infinite Mathematical Agent," (pp. 130 - 133) he describes how the Agent is tied to a semiotic system.

Rotman's "Subject" - the actual human mathematician - because they exceed the temporal and combinatorial limitations of unaided human faculties. But neither would these operations be the merely imagined actions of the "Agent." The boundaries between ideal and actual mathematical agency, between what can be done and what can only be imagined, are moving targets. They move as new agencies are introduced to mathematics.[180]

The way that the mathematician is "idealized" and the way that mathematical activity is imagined change through time depending on what metaphors or other cultural resources are at work. And the character of *actual* mathematicians and their *actual* activities change as well. The historicity of ideals and actuals in the character of mathematical agency resists the kind of ahistorical model that Rotman provides. Wang's work with the computer was intended to intervene at the boundary between what could only imagined to be done and what could actually be done. More, Wang read the computer as an improved approximation of a never-existing ideal mathematician. Seen this way, Wang's work also represents an intervention in the history of ideal and actual mathematicians and ideal and actual mathematical practice.[181]

In the rest of this chapter, I explore Wang's work in automated theorem-proving, focusing on the development of the Program P and its place in Wang's vision for a mathematics transformed by computation. I begin with an exploration of Wang's

[180]I would seek to historicize Rotman's "Agent" in part through the addition of perspectives from Actor Network Theory. Therein, knowledge-production agency is characterized as an emergent property of hybrid human-nonhuman networks rather than a stable faculty possessed once and for all in one way or another by any human or nonhuman entities alone. See, for example, Bruno Latour, *Reassembling the Social: An Introduction to Actor-Network Theory* (Oxford, U.K.: Oxford University Press, 2007).

[181]Amir Alexander has explored different mythologies and narratives that surround the character of the mathematician, in "From Voyagers to Martyrs: Toward a Storied History of Mathematics" in *Circles Disturbed: The Interplay of Mathematics and Narrative*, eds. Apostolos Doxiadis, Barry Mazur (Princeton, NJ: Princeton University Press, 2012): 1 - 51, on p. 41. See also, Amir Alexander, *Dual at Dawn: Heroes, Martyrs, and the Rise of Modern Mathematics* (Cambridge, MA: Harvard University Press, 2010).

professional career, and in particular on his complicated disciplinary affiliations - to mathematics, philosophy, and logic. He wanted recognition from all three and also sought to transform all three with computers in hand. Wang imagined a new research field within mathematics, called "Inferential Analysis" that would focus on the algorithmic study of mathematical problems in search of new, computationally informed insights. Inferential Analysis, of which automated theorem-proving would be a central component, was to bridge logic, computing, and mathematics, creating new possibilities for knowledge and practice in those areas. However, ultimately, his hybrid and transformative disciplinary perspective left him somewhat isolated from all three professional communities - standing in opposition to the abundant cases of "successful" interdisciplinarity that abounded in the 1950s. This disciplinary dimension of Wang's work in automated theorem-proving is the subject of the next section "Searching for Algorithms in the Branches of Mathematics." In "The Herbrand Universe," I explore Herbrand's Theorem and its intellectual context in more depth, pointing to those elements that were transformed by Wang's automation attempt. In "Reformalism I: Ruling Propositional Logic," I explore briefly the design and implementation of his first theorem-proving program, the System P, in order to contrast it directly with the Logic Theory Machine. But his more significant efforts were directed towards designing and implementing the Program P, discussed in "Reformalism II: Finding Patterns in the Predicate Calculus." That program represents his most concerted effort to realize his vision for mathematics, computing, and logic in the form of working programs. Throughout this dissertation I argue that implementation has epistemological stakes for the history of mathematics. This was certainly the case here. Wang took a theorem from proof theory and transformed the *kind of knowledge* it produced. It went from being an abstract statement about two branches of logic to a practical tool kit for proving theorems. And more, running the Program P yielded what Wang called a

"new and surprising discovery" *about Principia* - namely that every theorem from the predicate calculus contained in *Principia* has a particular formal structure, a result described in the conclusion of this chapter.

"Searching for Algorithms in the Branches of Mathematics"[182]

Wang was a native of China, and completed his first two academic degrees there during the years of the Second World War.[183] He received a B.Sc. in mathematics from the National Southwestern Associated University in China and a Master's in Philosophy from Tsing Hua University under the advisement of Prof. Shen Youding. Following Youding's lead, Wang elected to go to the United States for his doctoral studies in philosophy, and immigrated to Cambridge, MA in 1946 to study for the PhD at Harvard University under the advisement of Willard Quine. Although Wang became an American citizen and lived most of the rest of his life in the U.S., he carried strong allegiances

[182]This phrase is taken from Wang, "Computer Theorem Proving and Artificial Intelligence" p. 58, in which Wang describes his interest in exploring the potentially algorithmic dimension of mathematical problems.

[183]Or more specifically, what is known as the "War of Resistance" between China and Japan, between 1937 and 1945. See Wang, "From Kunming to New York," trans. Richard Jandovitz, Montgomery Link in *Hao Wang: Logician and Philosopher* [*Texts in Philosophy* 16] (London, UK: College Publications, 2011): 39 - 48. Unfortunately, a comprehensive biographical account of Wang's life is not possible here. More unfortunately, no comprehensive biography exists elsewhere. His collection at the Rockefeller Archive Center was only recently opened for research in the Spring of 2012. Some selective discussions of Hao Wang's life and translations of some of his early Chinese writings can be found in Charles Parson, Montgomery Link eds., *Hao Wang: Logician and Philosopher* [*Texts in Philosophy, Vol. 16*] (London, UK: College Publications, 2011). The volume was compiled before the opening of Wang's archive, and consists primarily of students' and colleagues reflections on Wang's life and work. Wang penned portions of an autobiography in Wang, *A Logical Journey: From Gödel to Philosophy* (Cambridge, MA: The MIT Press, 1996), esp. "Chapter Three: Religion and Philosophy as Guides to Action" and "Chapter Four: The Conversations and Their Background." My knowledge of Wang's life is also indebted to oral histories performed with Joyce Friedman, one of Wang's relatively few graduate students, and to Juliet Floyd, Philosophy Professor at Boston University, and Martin Davis, a colleague and fellow pioneer of automated theorem proving.

to China, to Chinese people, and to Chinese ideas throughout his life. Charles Parsons, a colleague of Wang's from Harvard Philosophy, suggested that "Although he became a US citizen in 1967, Wang would have resisted characterization as an Asian-American. I believe he thought of himself simply as Chinese, a member of the Chinese diaspora that has existed for centuries."[184]

Wang's philosophical program took many turns throughout his life. However, one conviction can be found at work in seemingly every corner of his intellectual career - Wang was deeply committed to *practical* philosophy. He wanted philosophy to be useful, to provide principles for action and daily living. This philosophical posture was one key motivation, I claim, for his interest in automated theorem-proving. No branch of philosophy was more isolated from action and utility than mathematical logic. Not even mathematicians found much use for the foundational formal systems logicians were so at pains to construct. By transforming an esoteric result of mathematical logic like Herbrand's Theorem into an actual actionable toolkit for theorem-proving, Wang

[184]Charles Parsons, "Hao Wang" in *Hao Wang, Logician and Philosopher*: 7 - 14, on p. 7. It will also be unfortunately beyond the scope of this discussion to engage the rich existing scholarship concerning the experiences of Chinese-American academics in the postwar United States. Interested readers should consult Zuoyue Wang, "Transnational Science during the Cold War: The Case of Chinese/American Scientists" in *Isis* Vol. 101, No. 2 (June 2010): 267 - 377. Zuoyue Wang (no relation) argues that it is a mistake (made all too often) to conflate Chinese science and scientists with Soviet science and scientists in the Cold War landscape. Zuoyue Wang explores both the work of Chinese scientists who studied in the United States and stayed there and Chinese scientists who worked or studied in the United States and then return to China as ways of complicating and enriching our understanding of the Americanization of international science and the Internationalizing of American science. See also Zuoyue Wang, "U.S.-China Scientific Exchange: A Case Study of State-Sponsored Scientific Internationalism during the Cold War and Beyond" in *Historical Studies in the Physical and Biological Sciences*, Vol. 30, No. 1 [Physicists in the Postwar Political Arena: Comparative Perspectives] (1999): 249 - 277. It is also worth noting that another of the significant early pioneers of Automated Theorem-Proving - Wu Wen-Tsuin - was Chinese - but remained in China. He and his body of work (not limited to theorem proving) were the subject of a recent dissertation: Jiri Hudecek, "You fight your way, I fight my way: Wu Wen-Tsun and Traditional Chinese Mathematics" Dissertation submitted to the Department of History and Philosophy of Science, University of Cambridge (November 2011). I am grateful to Jiri for many conversations about Wu Wen-Tsun's work and mechanical mathematics in China which has its own rich and fascinating history. It is not clear whether Wang and Wen-Tsun ever actually met but Wang was certainly aware of and impressed by his work.

hoped to make it a more useful result, as I will discuss later in this section.

There was a political dimension to Wang's emphasis on action as well. Throughout his work, Wang lamented the fact that philosophy was woefully distanced from the the lived experience and daily lives of human beings. This, in part, led him to develop what he described as an "infatuation" with Marxist philosophy after the *rapprochement* between the U.S. and China in the early 1970s. At that time, Wang saw an emphasis on action and utility in Marxism that he believed was lacking in much of Western philosophy. After his first visit to China, he reported "Although there was much [in Marxism] that puzzled me, I felt that I saw a comprehensive philosophy *unifying thought and action* in a way that I had not before considered possible."[185] That desire to unify thought and action, I propose, was also at work in his intention to make Herbrand's Theorem useful, to put it to work.

In the 1980s when the extent of the human rights abuses perpetrated under Mao Tse-Tung became visible to the world, Wang abandoned Marxism and began working on the development of his own philosophy of action, presented in its most fully developed form in a work aptly titled *Beyond Analytic Philosophy: Doing Justice to What We Know* (which Wang considered to be his mot important contribution to philosophy) in 1988. In it, he advocates for the role of common sense rather than completely formalized reasoning, in creating actionable knowledge about the world.[186] Marx remained an icon of the practical philosophy he wanted, however. In Wang's last (posthumous) publication he cites Marx's dictum that "The philosophers have only interpreted the world, in various ways; the point, however, is to change it," and argues that the true purpose of philosophy is to be a "guide to action."[187]

[185]Wang, "From Kunming to New York," p. 43, my emphasis.

[186]Wang, *Beyond Analytic Philosophy: Doing Justice to What We Know* (Cambridge, MA: The MIT Press, 1996 [1988]).

[187]Wang, "Religion and Philosophy as Guides to Action" in *A Logical Journey: From Gödel to Philosophy* (Cambridge, MA: The MIT Press, 1996): 101 - 127, p. 101.

It is difficult to ascertain what Wang's posture towards Marxism and Mao Tse-Tung's China was before the early 1970s, in no small part because of the prevalence of intense anti-communist sentiments and policies throughout American society at that time. We do know that, under his father's direction, Wang read certain materialist texts as a young boy.[188] We also know that as soon as it was possible, Wang visited China in the early 1970s and upon his return he became a very vocal advocate for Chinese interests and policies. Wang's colleague Martin Davis believed that Wang, in fact, became interested in computing in the first place because he wanted to develop a useful skill set and to "return to China and to participate in the work of the Communist regime."[189] At that time, he penned multiple "Letters to the Editor" that appeared in *The New York Times* and *The Washington Post* defending Mao and Chinese positions on foreign and domestic issues against American critics.[190] Upon Mao Tse-Tung's death in September of 1976, Wang even served as the chair of the memorial service at Hunter College in New York City, attended by upwards of 2000 people.[191]

Wang, like many academics from communist countries, did not escape the attention of the Federal Bureau of Investigation (FBI). On September 12, 1956 (while he was working on his theorem-proving programs) Wang was preparing to sail on the "Queen Mary" from New York to Southampton. He was en route to Oxford where he was then a lecturer after a summer of research at the Burroughs Corporation. The FBI

[188]He wrote: "During my middle school years, my father wanted me to read some books about dialectics and materialism, which at the time I felt I did not understand. Later, in the third year of high school, having read Professor Jin Yuelin's textbook, *Logic*, I felt mathematical logic was easy to understand. I thought if I first studied what was easy, later I might perhaps be able to understand what was difficult" in "From Kinming to New York," p. 43.

[189]See Davis, "Hao Wang's Contributions to Mechanized Deduction and to the Entscheidungsproblem" in *Hao Wang: Logician and Philosopher*: 73 - 78, p. 76.

[190]Wang's collection of newspaper clippings relating to Chinese affairs, and drafts and clippings of his editorial pieces are contained in RAC-RU-HW, Series 2 - Office Materials, Box 2, Folder 15.

[191]The program for this event, drafts of the key address delivered by Prof. Yang Chenning of Stony Brook University, and newspaper articles describing the event can be found in RAC-RUC-HW, Series 2 - Office Materials, Box 2, Folder 15.

was advised that Wang was "carrying nuclear physics instruments from US to England for delivery to Communist Chinese, Charge d'Affairs, and subsequent transmittal to Chinese mainland."[192] As such, Wang's luggage was searched upon boarding the ship and the following results reported:

> The subject [...] was in possession of Chinese Passport K714337 and was traveling as a stateless person under Affidavit in Lieu of Passport issued December 8, 1955. Customs representatives found a brown leather suitcase among subject's possessions which contained various electrical components which the AEC representative stated could be used in any device using electronic techniques, although he indicated that no special nuclear material was contained therein. [...] In searching subject's luggage, they also found three pamphlets entitled "Introduction to Marxism," "Soviet History of Philosophy" and "J. Stalin - Concerning Marxism in Linguistics."[193]

It turned out that Wang was transporting the computer components as a favor to Wen-Yu Chang, a visiting professor at Purdue University specializing in the study of cosmic rays. Chang was returning to China and feared he would not be permitted to bring the components with him because the FBI had already interviewed him as a suspicious person.[194] The image of Wang with a suitcase full of electronic components and Marxist pamphlets is perhaps aptly symbolic of the hybrid landscape through which Wang moved - from the technological tangibles of computing to the theories

[192]FOI/PA#1193350-0, FBI Report Form (10 October 1956). The report was unclassified on October 9, 2012 but the name of the source was redacted.

[193]FOI/PA#1193350-0, Letter to the Director of the FBI (September 13, 1956): p. 2. AEC stands for Atomic Energy Commission, the body responsible for the oversight of nuclear research and development following the Second World War. For more on the history of the AEC and programs of surveillance in the Cold War United States, see Alex Wellerstein, *Knowledge and the Bomb: Nuclear Secrecy in the United States* (PhD Dissertation: Harvard University, submitted October 2012), esp. "Information Control," "A Very Black Day", and "Peaceful Atoms, Dangerous Scientists", pp. 221 - 352.

[194]Chang indicated to the FBI he "had no place to go but to Communist China" because of "a lack of possibilities for advancement" at Purdue owing to a speech impediment. FOI/PA#1193350-0 Letter to the Director, (September 19, 1956), p. 3. It seems the FBI's suspicion about Chang was related to his connection with another subject of interest (whose name was redacted from FOIA documents) who was a member of the openly communist "Chinese Association of Scientific Workers" which was by then defunct. The electronics components were ultimately confiscated under the auspices that Wang failed to declare them, their value $1000.00 value exceeding the $25.00 allowance.

of practical political philosophy. Wang's work on automated theorem-proving might, at first pass, seem far afield from his later work on practical philosophy. However, I propose that both were representative of Wang's desire to unify thought and action, and to make philosophy *useful*, be it political philosophical or mathematical logic which occupied the early and the late decades of his career respectively.

As a doctoral student at Harvard, Wang's work his work remained firmly grounded in that abstract logical project inherited from Whitehead, Russell, and others in the early twentieth century to reduce mathematics to logic - to formalize mathematics in terms of the axiomatic and deductive structures of logic. His dissertation in particular set out to "reduce the classical arithmetic of natural and real numbers to a logistic system" - in particular one with weaker axioms and logical principles than had previously been given.[195] This project was very much in keeping with the tradition of analytic philosophy that Wang inherited from Youding, Quine, and the logicians who proceeded them, to understand branches of mathematics in terms of the formal structure of logical systems.

Upon graduation, Wang remained at Harvard first as a Junior Fellow with the Harvard Society of Fellows for the term between 1948 and 1951, and from 1951 until 1956 as Assistant Professor in the Department of Philosophy. During this time, Wang remained close with Quine and his work remained focused on the logical formalization of mathematics. However, beginning in the early 1950s, Wang was becoming increasingly dissatisfied with this work. In particular, he lamented the fact that analytic philosophy and logic were of little interest and relevance for the actual practices and interests of mathematicians. Logicians claimed that their studies revealed the foundations, justifications and the structure of mathematics - logic was not really of *use* or even of

[195]See Wang, "An economical ontology for classical arithmetic" Doctoral Dissertation submitted to the Department of Philosophy, Harvard University (1948) in *Harvard University Archive Collection*, HU 90.5483.2.

particular *interest* to many, if not most, practicing mathematicians. As I argued in the Introduction to this dissertation, Wang was dissatisfied *in general* with abstractions that could not be used or put to work in some fashion. He thought that philosophy in general should have relevance for human life, politics, and experience. And analytic philosophy should at least be relevant to practitioners of the discipline that it claimed to ground to illuminate.

Logicians aimed to provide a formalization of all branches of mathematics in terms of axiomatic, deductive systems - like that Whitehead and Russell sought to lay out in *Principia Mathematica*. The ultimate goal was to demonstrate that all proofs in mathematics could, at least in theory, be presented as comprehensive step by step deductions within such an axiomatic system. However, mathematicians seldom actually worked this way. Mathematical proofs seldom consist of exhaustive inferential deductions. Wang lamented this disconnect between mathematical practice and logical formalization: "A common complaint among mathematicians is that logicians, when engaged in formalization, are largely concerned with hairsplitting. It is sufficient to know that proofs can be formalized. Why should one take all the trouble to show exactly how such formalizations are to be done, or even to carry out actual formalizations?"[196] Wang wanted logic to be more relevant to mathematicians and more useful to the production of mathematical knowledge.

Wang believed that computers could be used to forge such connections between mathematics and logic. In his own telling, it was dissatisfaction with analytic philosophy, or rather "philosophy (as seen at Harvard)" that he became interested in computers in the first place.[197] In 1953 and 1954, he took his first foray out of traditional academic institutions when he accepted an appointment as a "research engineer" at the

[196]Wang, "Towards Mechanical Mathematics," p. 18.
[197]Wang, "Computer Theorem-Proving and Artificial Intelligence," p.50.

Burroughs Corporation research center in Paoli, PA. At the time, however, the Burroughs Corporation center housed only one computing machine and Wang was never able to use it, and he reports being "discouraged from taking the course for electronic technicians."[198] As such, Wang's first attempt to "work with computers" was in fact another study in abstract formalism pursued on that more familiar medium - paper. During 1953, he in fact crafted a formally equivalent alternative to Alan Turing's more famous "notional machine," usually called a Turing Machine. Wang designed an equivalent machine that he believed to operate according to more basic principles.[199] His experience at Burroughs unfortunately left him in the world of paper, without either a technical programming skill set or the tools to forge a bridge between logic and mathematics.

After this first stretch of time in the United States, Wang began a period of overlapping and mixed institutional affiliations. In 1955, he was appointed the John Locke Lecturer in Philosophy at Oxford University, and he kept an Oxford affiliation as a Reader in the Philosophy of Mathematics at Oxford from 1956 to 1961. During that last interval, Wang travelled back to the United States most summers to hold visiting researcher and consultant positions at two other industrial research institutions - IBM Research Labs in New York during the summers of 1956 and 1957 and Bell Telephone Research Laboratories, 1959 - 1960.[200] It was during these later visits at IBM and

[198]Wang, "Computer Theorem-Proving and Artificial Intelligence," p. 50.

[199]This work was not published until some time later, in 1957. See Wang, "A Variant to Turing's Theory of Computing Machines" in *Journal of the Association for Computing Machinery*, Vol. 4, No. 1 (1957): 63 - 92. In spite of not having access to the computer while at Burroughs, Wang also appears to have worked on the question of "error prevention" in computing: he considered the kinds of errors that can befall computers and engaged some of John von Neumann's work on making computers more reliable than each component. Wang only produced one report on this subject, "The Control of Errors," Technical Memorandum No., 54-59, Project No. R5781 (RAC-RU-HW, Series 3, Box 37, Folder - "John von Neumann", June 8, 1954).

[200]Some of Wang's movements and affiliations in this time are difficult to identify firmly and his institutional affiliations have been largely glossed over and over-simplified in what little literature exists that surveys his life and work. My reconstruction of his career trajectory is based on several mini-biographies and c.v.'s that Wang circulated to different institutions and that remain in his archive

Bell Labs that Wang finally got his hands on modern digital computing machinery - specifically IBM System\360s - and went to work trying to design and implement theorem-proving computer programs for the propositional and predicate calculus.

Two of the three papers in which he details his results were published in the journals of those industry research centers: "Toward Mechanical Mathematics" was published in the *IBM Research Journal* and "Proving Theorems by Pattern Recognition - II" was published in the the *Bell Systems Technical Journal*. The third article, "Proving Theorems by Pattern Recognition I" was published in the Journal of the *Communications of the Association for Computing Machinery*. Although Wang had set out to forge a bridge between logic and mathematics, his work found its home primarily in technical computing presses. This might not have troubled Wang, and indeed many mathematicians requested preprints of these articles, but it did point to the fact that Wang's work in automated theorem-proving occupied a strange disciplinary territory at the intersection of logic, mathematics, engineering, computing, and analytic philosophy. This of course was nothing at all new in the 1950s - interdisciplinary and trans-disciplinary work was becoming increasingly common during the 1950s, especially where computers were present. However, most inter-disciplinary stories that are told in history of science are success stories.[201] While it is certainly the case that American science took on an increasingly collaborative, large scale, and interdisciplinary char-

(though even they contain slightly different pieces of information) including Wang, Letter to David Smith (12 January 1967) in RAC-RU-HW, Series 2, Box 8, Folder 163; Wang, Award Citation, (1955 - 1956), RAC-RU-HW, Series 2, Box 18, Folder 443; Wang to the Marian Lucius, Rockefeller University Registrar (c.a. 1980) in RAC-RU-HW, Series 1, Box 1, Folder 2.

[201]The rise of interdisciplinary work as a new paradigm for scientific research in the postwar United States is discussed, for example, in Stuart Leslie, *The Cold War and American Science* (New York, NY: Columbia University Press, 1993); Peter Galison, Bruce Hevly, eds. *Big Science: The Growth of Large Scale Research* (Sanford, CA: Stanford University Press, 1994). Peter Galison has emphasized elsewhere that interdisciplinarity was not easy, especially where communication practices needed to forged across communities with very different training and commitments. However, he also points to the reorganization of postwar science around larger scale and less disciplinarily bounded efforts throughout the period. See Galison, "Trading Zone: Coordinating Action and Belief" in *The Science Studies Reader*, ed. Mario Biagioli (New York, NY: Routledge, 1999): 137 - 160.

acter throughout the postwar period, not every attempt to build bridges and forge cross disciplinary interest and collaboration was successful. Wang's work, and perhaps automated theorem-proving as a whole might be seen as less successful examples of this work. This may be in large part because as a community, researchers in this field continuously struggled to garner the interest of practicing pure mathematicians.

In 1961, on the heels of his work in automated theorem-proving, Wang returned to Harvard University. Wang's negotiations with then Dean Harvey Brooks between 1960 and 1961 concerning the his appointment, and the title of his appointment are surprisingly indicative of the disciplinary terrain that Wang traversed and sought to reorganize. Wang was offered a faculty position with the Harvard Division of Applied Science. The Division would sponsor half of his appointment, and after his arrival, Wang would have the opportunity to establish an affiliation with either Mathematics or Philosophy to account for the other half.[202] It was clear that Wang was enthusiastic about having an affiliation with the Mathematics department, writing that "I should be very happy to be a member of the mathematics faculty, if such an arrangement would be satisfactory to the mathematics department."[203] When Brooks first offered Wang the position, it came with a particular title: "I am authorized by Dean Bundy to offer you officially an appointment at Harvard as *Professor of Pure and Applied Mathematics*."[204]

However, it became clear that without an official affiliation with the Harvard Mathematics Department "pure" could not be included in his title. Brooks wrote "After discussion with the President, we decided to put through your appointment as *Professor*

[202]In the meantime, the second half of his salary would be furnished by the general funds of the Faculty of Arts and Sciences. See Letter from Harvey Brooks to Wang, 23 December 1960 (RAC-RU-HW, " Series 2, Box 5, Folder 97 "Brooks etc. 1960 - 1961").

[203]Letter from Wang to Brooks 14 January 1961 (RAC-RU-HW, Series 2, Box 5, Folder 97 "Brooks etc. 1960 - 1961").

[204]Letter from Brooks to Wang, 23 December 1960 (RAC-RU-HW, Series 2, Box 5, Folder 97 "Brooks etc. 1960 - 1961"), my emphasis.

of Applied Mathematics with the question of the "*Pure Mathematics*" in the title left open until it is finally settled after you are here whether your "other half" should be in Mathematics or Philosophy."[205] The domain of "pure mathematics" as negotiated at Harvard was reserved for the Mathematics department and Wang would have to find a place for his divided self there if he would have that title. Brooks couched the change as a way of keeping things open until Wang could decide "on the scene" how his affiliation should look. Wang took offense. He truly felt that the work he was in part "pure mathematics" in spite of its applied dimensions. He responded:

> I am surprised and feel very embarrassed by the suggested change of the originally proposed title. [...] Since, however, this has already come up, I think I ought to be entirely candid about my own views on this matter. It is my belief that the slightest misunderstanding or misgiving at the beginning tends to poison relations in the long run, and my somewhat unorthodox interests add complexity to matters of the kind. In all sincerity, one of the greatest attractions of Harvard for me is the apparent willingness to encourage new experiments and new combinations of different fields [...]. As a temporary solution, I am not able to see the advantage of "Applied Mathematics" over "Pure and Applied Mathematics": in either case, the title will presumably have to change if the second department turns out to be philosophy. Would perhaps something like "Professor of Mathematical Logic" evade all the difficulties and represent my own work more faithfully?[206]

The negotiations continued. Brooks suggested "Mathematical Logic and Applied Mathematics," to which Wang counter-proposed "Pure and Applied Logic" or "Pure and Applied Mathematical Logic," indicating that for Wang, the descriptor "pure" grabbed a hold of part of his work that he was hesitant to forfeit. Eventually, he agreed to the title of "Mathematical Logic and Applied Mathematics."

While the exchange might be disregarded as a superficial or prideful negotiation, and pride was perhaps part of it, it also signals two very crucial elements of Wang's

[205] Letter from Brooks to Wang, 11 March 1961 (RAC-RU-HW, Series 2, Box 5, Folder 97 "Brooks etc. 1960 - 1961"), my emphasis.

[206] Letter from Wang to Brooks, 16 March 1961 (RAC-RU-HW, "Hao Wang Papers," Series 2, Box 5, Folder 97 "Brooks etc. 1960 - 1961")

work and experience. Wang thought his work was *as much* pure as it was applied. It concerned not just theory or application but instead *how application could contribute to pure mathematics.* The second thing signaled by this exchange is that the words "pure" and "applied" were not free floating labels but were rather tethered to particular departments and institutional contexts. "Pure" didn't designate content alone but also profession and affiliation. Wang conceded: "I have to admit that initially the offer of an elegant and conventional title came as a pleasant surprise and I was not clear about the relation of the University to its various departments on deciding such a matter."[207] Wang subscribed to one particular characterization of how the computer would intervene in mathematics, and how it should be studied. He didn't advocate (as many other automated-theorem proving practitioners did) for a discipline and associated department for "computer science" but rather wanted to work within and around and across existing disciplines.

Wang ultimately accepted the title "Mathematical Logic and Applied Mathematics" - which he held, unaltered, until 1967 when he left Harvard University for good. The second half of his salary continued to be paid by the "general fund" of the Faculty of Arts and Sciences, indicating that no official secondary affiliation - with either Philosophy or Mathematics - was ever established. When he left, Wang's concerns were in large part financial.[208] However, Wang's concern about departmental affiliation remained a central issue for his professional decisions. After receiving a definitive offer from the University of Pennsylvania, and what Wang termed "definite feelers" from Columbia and Stanford, he wrote the following to Dean Brooks: "I believe there

[207]Letter from Wang to Brooks, 11 April 1961 (RAC-RU-HW, Series 2, Box 5, Folder 97 "Brooks etc. 1960 - 1961").

[208]After extensive correspondence about the possibility of his leaving through 1967, Wang drafted a note called "Summary of Issues" that he sent to Dean Brooks, highlighting what he perceived as unjust compensation relative to his colleagues and to offers he had received from other universities, especially the University of Pennsylvania. Wang, "Summary of Issues," December 13, 1965 (RAC-RU-HW, Series 2, Box 30, Folder 756).

are several first-rate universities where I can get distinctly better paid positions than my present one at Harvard. In fact, I can command such better positions in several departments (philosophy, mathematics, computer science)."[209] And when he left Harvard University, he chose to accept a position at the Rockefeller University in which he would direct his own "Logic Group" that would consist of more and less permanent affiliates from mathematics, philosophy, logic, and computer science (which was emerging as a discipline in its own right throughout the 1960s). Wang's Logic Group experienced moments of excitement with logicians and philosophers like Saul Kripke, Robert Solovay, Ariel Levy visiting for various terms. And at other times, Wang was the sole member of the group. He retired in 1991, never having been affiliated with a mathematics department (excepting the one from which he received his B.Sc. in China), and having largely retreated from dominant conversations in logic and philosophy. He invested a great deal of time in the later years of his career interviewing mathematician and Kurt Gödel on his views about mathematics, minds, computers, and philosophy. Wang is best known outside of automated theorem-proving for his presentations and interpretations of Gödel's work and philosophy.[210]

Throughout his career, Wang was at the same time a member of elite communities and institutions in mathematics, philosophy, and logic and at the same time was, often by his own design, operating outside of them. He recognized that his vision for automated theorem-proving didn't actually fit into mathematics, engineering, logic, or philosophy as they existed in the late 1950s. Instead, he proposed the creation of a new field of research that he called "inferential analysis" that would involve a "novel combination of psychology, logic, mathematics, and technology."[211] The field

[209]Wang to Harvey Brooks, November 30, 1965 (RAC-RU-HW, Series 2, Box 30, Folder 756), p. 2.

[210]In particular see Wang, *Reflections on Kurt Gödel* (Cambridge, MA: The MIT Press, 1987) and Wang, *A Logic Journey: From Gödel to Philosophy* (Cambridge, MA: The MIT Press, 1996).

[211]Wang, "Computer Theorem and Artificial Intelligence," p. 56. He first introduced the idea in "Toward Mechanical Mathematics" but some of the preliminary ideas can be found in his earlier piece

would serve to institutionalize the the transformational effect Wang believed computing could offer to mathematics and straddle boundaries between mathematics, logic, and computing and the theories and applications within them.

Rather than focusing on solving specific problems, Wang wanted a field concerned with a kind of meta-problem-solving - "inferential analysis" would aim at the exploration and formulation of general processes of problem solving. Wang imagined inferential analysis as a counter-part to the existing field of numerical analysis. Where the latter aimed to unpack the structures and processes of numerical calculation, the former would aim to study structural properties of nonnumeric problems and to develop methods of inference for solving them: "In contrast with pure logic, the chief emphasis of inferential analysis is on the efficiency of algorithms, which is usually attained by paying a great deal more attention to the detailed structures of the problems and their solutions, to take advantage of possible systematic short cuts."[212] The field included a theoretical dimension of exploring the structural properties of problems and solutions but also a practical dimension of how to exploit those structures with feasible computation. It was both an *application* of pure logic, but had the ability to in turn contribute novel theoretical insights to logic and mathematics by making visible new structural and algorithmic dimensions of mathematical objects and problems. Wang taught some courses on "inferential analysis" at Harvard University, but seldom referenced it after the mid-1960s. Wang wanted a new field of mathematics that aimed to think about proof through the lens of mechanization. This field never really emerged outside of his immediate circles. Instead of a new mathematical field, a new discipline emerged - computer science - that contained elements of "inferential analysis" in its formal study of algorithms, but that was not focused on the work of proof. Wang never had an affil-

Wang, "A Variant to Turing's Theory of Computing Machines" in *Journal of the Association for Computing Machinery* Vol. 4, No 1 (1957): 63 - 92.

[212]Wang, "Proving Theorems by Pattern Recognition I," p. 220.

iation with a computer science department, and in fact, never really worked hands-on with computers after the 1960s.

However, in 1983 Wang received, much to his surprise, more formal recognition for his work in automated theorem-proving, and some recognition in mathematics. He received the first ever "Milestone Award in Automated Theorem-Proving." The award was furnished by the American Mathematical Society, and was awarded at an AMS conference that included sessions in automated theorem-proving. The goal of the conference was to make the work of automated theorem-proving practitioners visible to practicing mathematicians, and to cultivate collaborations and interest across those communities. By 1982 certain significant results, like the computer-assisted proof of the infamous Four Color Conjecture had garnered some interest in the subject. The early 1980s also saw an effort on the part of automated theorem-proving practitioners to solidify their field, including the creation of a dedicated journal - the *Journal of Automated Reasoning*, the creation of the *Association of Automated Reasoning* and the publication of several consolidating and synthesizing volumes of related articles.[213] Wang's work was widely recognized in those "disciplining" moments as inaugural and significant, in spite of the fact that many of Wang's disciplinary and departmental struggles. I turn now to that work that won Wang the award in order to see what Reformalism transpired in Wang's implementation of Herbrand's Theorem.

[213]Including J. Siekman, G. Wrightson, eds. *The Automation of Reasoning: Classical Papers on Computational Logic, Volumes I and II*, (Berlin: Springer, 1983) and D. Loveland, ed. *Automated Theorem Proving: After 25 Years*.

The Herbrand Universe

A substantial amount of early-twentieth century logical inquiry was directed towards what is called the "decision problem," or the *Entscheidungsproblem*.[214] It was originally introduced in *Grundzüge der Theoretischen Logik* (Principles of Mathematical Logic) - a 1928 text by German logicians David Hilbert and Wilhelm Ackermann. Hilbert and Ackermann were building directly on the project laid out by Whitehead and Russell, namely the "complete enumeration of all the ideas and steps in reasoning employed in mathematics."[215] Hilbert and Ackermann also sought to completely formalize the branches of mathematics as axiomatic systems.[216] These could in turn be analyzed relative to certain key formal properties that correlate to what kinds of problems can (or can't) be solved in that system and how they can (or can't) be solved.[217] One such formal property was "decidability."

Logicians would call a system decidable if they can design a procedure that can take any statement from that system and, in a finite number of steps, return an answer of "yes" or "no." The "yes" or "no" outcomes indicate whether or not that particular statement was "universally valid" - no matter what interpretations are assigned to the

[214]Accessible introductions to the decision problem can be found in Petzhold, "*Das Entscheidungsproblem*" in *The Annotated Turing: A guided Tour through Alan Turing's Historic Paper on Computability and the Turing Machine* (Indianapolis, IN: Wiley Press, 2008):199 - 322; Martin Davis, *Computability and Unsolvability* (New York, NY: Dover, 1973 [1958]), esp. xv - xvii, 66 - 80, and 162 - 178; Stephen Kleene, "Computability and Decidability" in *Mathematical Logic* (New York, NY: Dover, 1967): 223 - 282.

[215]Whitehead, Russell, *Principia Mathematica*, Volume 1 p. 3.

[216]Hilbert is perhaps the person most strongly affiliated with this goal. Hilbert was committed to the "axiomatic" approach to mathematical foundations. In 1921, he laid out at program for the future of mathematics, usually called "Hilbert's Program" that had two parts: the program "calls for a formalization of all of mathematics in axiomatic form, together with a proof that this axiomatization of mathematics is consistent." See Richard Zach, "Hilbert's Program" in the *Stanford Encyclopedia of Philosophy*, 2003 (http://plato.stanford.edu/archives/spr2009/entries/hilbert-program/), p. 1. For a historical exploration of Hilbert's Program see Grattan-Guinness, *The Search for Mathematical Roots, 1870 - 1940* (Princeton, NJ: Princeton University Press, 2000): 470 - 474.

[217]Among them, "consistency" - an axiomatic system was consistent if it contained no contradictions (i.e. no statement and its negation could both be true); "completeness" - an axiomatic system was complete if all statements that were true within it could also be proved within it.

constituent elements of the statement, it will be true.[218] These "decision procedures" were abstract algorithms that would verify if any statement from an axiomatic system was a tautology or not. Some decision procedures were found quite easily by logicians in the 1920s. Others proved more difficult. Hilbert and Ackermann presented a decision procedure for the propositional calculus in their 1928 treatise and they also posed a challenge: to find a decision procedure for the predicate calculus:

> While the decision problem was easy to solve in the sentential [propositional] calculus [...] in the predicate calculus it presents a very difficult problem which as a whole remains unsolved. [...] Because of the central position of the problem, however, even the attempts to give a decision procedure at least for as large a class of formulas as possible are of great interest.[219]

Hilbert and Ackermann suggested some directions a solution might take, but ultimately, the decision problem for the predicate calculus remained unsolved until 1936.

And in fact, the task laid out by Hilbert and Ackermann turned out to be impossible. No decision procedure can be constructed that would always correctly return "yes" or "no" after a finite number of steps when given any statement that could be constructed in the predicate calculus. Any algorithm that could be constructed would fail (i.e. it would never stop, never return either "yes" or "no") when run on certain predicate statements. The undecidability of the predicate calculus was proved independently by two logicians in 1936 - Alan Turing and Alonzo Church.[220]

[218]See David Hilbert, William Ackermann, *Grundzüge Der Theoretischen Logik* (Berlin: Springer Verlan, 1928), pp. 72 - 81. The decision problem can be formulated for the question of "validity," "satisfiability," "provability," and "refutability" each requiring an algorithm to answer a different but equivalent "yes" or "no" question (equivalent meaning that for any statement the answer to all questions will either be "yes" or "no.") The formulations pick out different properties of axiomatic systems, but they are all equivalent. Hilbert and Ackermann give both an exposition of both the decision problem for validity and for satisfiability.

[219]Hilbert, Ackermann, *Principles of Mathematical Logic*, trans. L. Hammond, G. Keckie, R. Steinhardt (New York, NY: Chelsea Publishing Company, 1950 [1928]), p. 117.

[220]Alan Turing, "On Computable Numbers with an Application to the *Entscheidungsproblem*" in *Proceedings of the London Mathematical Society*, Vol. 42 (1937) [prepared, received in 1936]: 230 -

The negative result of the decision problem for predicate calculus cast a shadow on Hilbert's project (and, of course, that of Whitehead and Russell before him). More damaging were Kurt Gödel's "incompleteness" results from 1931 in which he demonstrates that sufficiently complex axiomatic systems must either be inconsistent or incomplete. Meaning, either they will contain contradictions (i.e. a statement and its negation will both be provable within it) or there will be statements that are true within the system that cannot be proved within it. The goal of reducing mathematics to formal logical structures appeared increasingly infeasible as the limitations and constraints of those formal structures were revealed.[221] Mathematics turned out to be more complicated than many mathematical logicians had hoped - not all of its truths lay dormant in axioms waiting for rote mechanical deduction to reveal them by an always terminating, finite number of iterative steps.

These negative results, however, did not empty mathematical logic of its interest and relevance. Although it appeared that no logical system would ever be able to grab a hold of all of mathematics, many interesting questions remained and new questions emerged. If not all axiomatic systems are decidable, which ones are and which ones aren't? What relationships obtain between those that are and those that aren't? What other properties might be formally correlated with decidability and undecidability? Herbrand's Theorem, upon which Wang's Program P was based, was most relevant with regards to this kind of question.

Jaques Herbrand was a French mathematician born 1908. He presented the theorem that bears his name is his PhD dissertation, "Recherches sur la Théorie de la Démonstration" one year before his premature death in a mountaineering accident.

265; Alonzo Church, "An unsolvable problem of elemental number theory" in *American Journal of Mathematics*, Vol. 58 (1936): pp. 345 - 363; Church, "A Note on the Entscheidungsproblem," in *Journal of Symbolic Logic*, Vol. 1 (1936): 40 - 41.

[221]See Gödel, "Über formal unentscheidbare Sätze der *Principia Mathematica* und verwandter Systeme, I," in *Monatschefte für Mathematik und Physik*, Vol. 38 (1931): 173 - 198.

He received his degree from the Université de Paris and worked under the advisement of French Logician Ernest Vissiot. Herbrand was also inspired and motivated by Whitehead and Russell's approach to reducing mathematics to logic and he was fully committed to Hilbert's Program of full and finitist axiomatizations of mathematics.[222]

Herbrand's Theorem established that a particular relationship obtains between the propositional calculus and the predicate calculus. The propositional calculus is the simpler of the two, consisting of variables x, y, z, ... and the logical operators "AND" ($\wedge$), "OR" ($\vee$), "IMPLIES" ($\rightarrow$) "NOT" ($\sim$), with "EQUIVALENCE" ($\equiv$). For example, $(x \vee y) \rightarrow z$ (read as "'x or y' implies z), $x \rightarrow (x \wedge y)$, and $z \vee (x \wedge (y \rightarrow z))$ are all propositions in propositional logic. The predicate calculus is more complicated. In addition to variables and the basic logical operators, it includes what are called "quantifiers" - "FOR ALL" ($\forall$) and "THERE EXISTS" ($\exists$) which enable the construction of more complicated propositions. For example, $\forall x \exists y : (x \rightarrow \sim y)$ (read as "for all x, there exists some y such that x implies not-y"), $\exists a, y : (y \wedge x) \rightarrow \sim z$, and $\forall x : x \rightarrow y$ are all statements in the predicate calculus.

Another crucial difference between the predicate calculus and the propositional calculus concerns the kinds of values that can be assigned to the variables they contain. The propositional calculus contains only what are called "logical variables" - they can only be assigned the values of True or False. A logical proposition is a "theorem" if it is a tautology: if it is rendered "true" by any assignment of True or False to its variables. For example $x \vee \sim x$ is a tautology because either x or its negation is always true. Conversely, $x \wedge \sim x$ is never true because x and its negation can never both be true.

The predicate calculus, on the other hand, also contains non-logical variables that

[222]For a discussion of Herbrand's training and influences see Warren Goldfarb, "Introduction" in *Jacques Herbrand, Logical Writings*, ed. Warren Goldfarb (Dortrecht, Holland: Reidel Publishing, 1971): 1 - 20 and Claude Chevalley, Albert Lautman, "Biographical Note on Jacques Herbrand" in *Jacques Herbrand, Logical Writings*: 21 - 23.

can be assigned values from different branches of mathematics. This is what made the predicate calculus a candidate foundation for mathematics. You could have variables that represent the integers, for example or that represent functions, and so on. Statements in the predicate calculus are true if the assignment of values those non-logical variables renders the statement true. So for example, if x, y, and z represent integers I can construct a predicate statement like $\forall x \exists y : (x + y) > 10$.This statement is a theorem. No matter what integer value I assign to x, I can add some number to it to get a number greater than 10. That predicate calculus includes non-logical variables is part of what makes it more complicated and more powerful than the propositional calculus.

Herbrand's goal was to find a way to formulate the predicate calculus in terms of the propositional calculus - the more complicated domain in terms of the simpler one. In particular, he wanted to show how theorems in the predicate calculus might be *proved* using the propositional calculus. Any successes in reducing the predicate calculus to formulations in the propositional calculus would be surprising given that the propositional calculus is *decidable* and the predicate calculus is not - solving problems in the predicate calculus is *harder* than solving problems in the propositional calculus. But Herbrand's Theorem offered some success in this regard: it established a particular relationship between the two branches of logic.

Herbrand's Theorem provides a method for proving theorems in the predicate calculus using the propositional calculus. It states that for any statement P in the predicate calculus, it is possible to construct a corresponding infinite series of statements in the propositional calculus, $S_1, S_2, S_3, \ldots$. The construction of this corresponding series is somethings called the "elimination of quantifiers" since P can contain the quantifiers "for all" and "there exists" where as no S_n in the propositional calculus can. This is also a process of "reduction" since it is reducing an artifact from a more complicated domain

to an artifact in a simpler one. Herbrand's Theorem goes on to show that $S_1, S_2, S_3, \ldots$ will be such that P is a theorem if, and only if, there exists a number N such that $S_1 \vee S_2 \vee \cdots \vee S_N$ is a tautology, where $\vee$ is the "OR" operator. Such a series is called a "disjunctive series" because the OR operator is also called the "disjunctive" operator. That is to say if for some N, the disjunction $S_1, \vee S_2 \vee \cdots \vee S_N$ is true no matter what values are assigned to the variables contained in each S_i, then P is a theorem. The theorem presented the possibility of proving theorems in one logic using the elements of another simpler one.

Of course, you can never get something for nothing. It was not possible to completely get around the fact that predicate calculus is not decidable - that it is intrinsically more complicated than the propositional calculus. Herbrand's Theorem suggested an "in principle" method for searching for a proof of P. One way would be to test if $S_1 \vee S_2 \vee \cdots \vee S_N$ is a tautology for $N = 1, 2, 3 \ldots$. If P *is* a theorem in the predicate calculus, then eventually this iterative search will produce a tautology and this would suffice as a proof of P. This may sound easy enough, but even for fairly simple predicate statements, N is usually an astronomically large number. But worse, if P was not a theorem, this process would never stop - you would just keep on looking for a tautology forever that you were never going to find.

So Herbrand's Theorem offered an interesting insight to the relationship between predicate calculus and propositional calculus, between quantified statements and quantifier-free statements. This relationship became even more interesting in 1936 when Turing and Church proved that the predicate calculus was not decidable: it also offered a relationship between a decidable and an undecidable domain of logic. But the theorem did not offer a practical procedure for *actually* proving theorems from predicate calculus.

Nor was it intended to. Virtually *all* decision procedures were impossible to put into practice. Even in cases where the procedure would work, there were too many cases for

a person to execute. These and similar procedures were often described in the language of machines and mechanical operations, but these too were only imagined. They were "abstract" machines, "notional" machines that could be described but not actualized. In his proof of the undecidability of the predicate calculus, Turing crafted one of the most famous such "notional machines" - the Universal Turing Machine - to demonstrate the existence of statements in that logic for which any finite algorithm would fail to return "yes" or "no," "universally valid" or "not universally valid." And as Charles Petzhold put it, "Turing's imaginary computers have unlimited storage and all the time in the world, so Turing can journey where the machine-bound programmers fear to tread."[223] Imaginary and abstract machines were the bread and butter of mathematical logic in the 1920s and 30s but these were about to meet their materialized match.[224]

As described in the previous sections, Wang was a student of this logical tradition. He was as familiar as one can be with the work of Hilbert, Herbrand, and the many others working to reduce mathematics to logic and to study the formal properties of axiomatic systems. And Wang lamented the fact that this work was not *useful* for actually proving theorems on the ground and he lamented the fact that practicing mathematicians were therefore not particularly interested in logic and its results. He thought that actual computing machines of the kind that were developed in the 1940s created new possibilities for setting the inert abstract procedures of logic in action.

Computers would scarcely be able to use Herbrand's method any more than a person could. Especially in the early decades of their development, computer memory would be insufficient to handle the *trillions* of cases that could be required to find the desired tautology that would prove some P to be a theorem. And when P was not

[223]Petzhold, *The Annotated Turing*, p. 220.

[224]Hao Wang himself later compiled two appendices in which he describes several of the most significant such "abstract machines" See Wang, "Appendix B: Algorithms and Machines" and "Appendix C: Abstract Machines" in *Popular Lectures on Mathematical Logic* (New York, NY: Litton Educational Publishing, 1981): 198 - 270.

a theorem, the computer was as lost as a person - it would go on checking increasing values of N until it ran completely out of resources or until it was stopped by a human operator. If Wang wanted to use Herbrand's Theorem to design a running program to prove theorems from *Principia Mathematica*, he had to transform the method into something that a computer could handle.

Reformalism I: Ruling the Propositional Calculus

Wang published the results of his first attempt to design and implement a theorem-proving program in "Toward Mechanical Mathematics." Here he first published his vision of "inferential analysis" and some of his optimistic views for the role that computers might play in future mathematics. He also gave a description of his first theorem-proving program, the "System P" that proved theorems from the propositional calculus - that more simple branch of logic. Before moving on to look at Wang's implementation of Herbrand's Theorem in the Program P (the focus of the next section), it is worth pausing for a brief comparison with the Logic Theory Machine. The latter program was also capable only of tackling propositional logical theorems from the first three chapters of *Principia*. In the previous chapter, I explored how Newell and Shaw designed a new programming language for the JOHNNIAC computers - the Information Processing Language - consisting for the most part in "list processing operations" for creating, manipulating, and analyzing linked list data structures. They translated the "rules of inference" laid out in *Principia Mathematica* - as interpreted by Hilbert and Ackermann - into forty-four variations of eight basic list-processing operations.[225] I argued that this was a *reformalism* of *Principia*, introducing new formal and material

[225]Both Wang and Newell-Simon drew from the rules of inference as laid out by Hilbert and Ackermann rather than as presented in *Principia* itself. See Hilbert, Ackermann, *Principles of Mathematical Logic*, esp. §§ 3 -9 in which they "establish the following rules for the transformation of logical expressions".

manifestations of its logic and new practices and processes for proving theorems.

Wang also enacted a *reformalism* of *Principia* in developing and implementing his System P. However, his translation looked quite different. His program was based on an insight from the work of Herbrand and another German logician Gerhard Gentzen: any statement in the predicate calculus can be broken down into its simplest constituent parts such that if the simpler parts are theorems, then the original statement will be as well.[226] Using this insight, Wang created a set of eleven rules for pulling a given logical proposition apart into sets of simpler and simpler formulae (ones that had fewer and fewer logical connectives) until all that was left is set of atomic formulae that can't be further taken apart.[227] Once the proposition had been so broken down, the computer needed only to check if a certain property held among those atomic formulae and if so, the original statement was a theorem.[228] He provides a formal statement of those rules, but he *also* described how those rules would be executed step by step by the IBM 701 computer.

[226]Wang cites the result in "Towards Mechanical Mathematics." p. 4. The result is originally presented in Gentzen, "Untersuchungen über das Logische Schliessen," in *Mathematik Zeitschrift*, Vol. 39 (1934-1935) 176 - 210, 405 - 431.

[227]The rules were used to iteratively "pull apart" any statement from the propositional calculus by introducing and removing logical operators such that after each iteration, the proposition would be left with one fewer connective, until only so-called atomic formulae were left. Atomic formulae here are the basic unit out of which complicated logical propositions are constructed. They are indivisible, in that if you removed any piece of them, they would cease to be sensical (or well-formed) propositions at all. Think of them as representing propositions like P = "the sky is blue" or Q = "it is raining." There are rules for building more complex formulas out of these atomic ones like P IMPLES (NOT-Q) here would mean "The sky is blue" implies that "it is not raining." Of course, these natural language sentences are just interpretations of the formal system that can be studied independent of any such semantic content.

[228]There is one fundamental rule that Wang called *P1* that states that "if λ, ξ are strings of atomic formulae, then $\lambda \rightarrow \xi$ is a theorem if some atomic formula occurs on both sides of the arrow." The arrow in this statement is not the typical "IMPLES" logical arrow, but is rather called a "sequent arrow." It is part of the formalization that Wang gives to the propositional calculus in order to design the System P that is amenable to being "pulled apart" in this way. In his formulation, every statement that can be input to the program will contain a sequent arrow. The rules for "pulling apart" the statement act on both sides of the arrow such that, when they can't be pulled apart any more, if the same atomic formula appears on both sides of that arrow, then the original statement is a theorem. Those interested in the formal specification of this system and the eleven rules can consult Wang, "Toward Mechanical Mathematics" pp. 4 - 6. The implementation is given on pp. 6 - 7.

In Wang's implementation, every logical proposition is input to the computer as a concatenation of seventy-two symbols (the number of columns on an IBM 704 punched card), each to be stored in its own address in memory. He implemented each of his eleven rules in terms of how it would operate on the seventy two memory addresses for a given proposition in question. For example, the very beginning of any run of the program would proceed as follows:

> The program enables the machine to proceed as follows. Copy the card into the reserved core storage COL1 to COL72 (72 addresses in all) in the standard BCD notation, i.e., a conventional way of representing symbols by numbers, one symbol in each address. Append the number 1 at the last address, viz., COL 72. Search for the arrow sign. If it does not occur, then the line is regarded as ordinary prose, printed out without comment, and the machine begins to study the next card. In particular, the machine stops if the card is blank. If the arrow sign occurs, then the machine marks all symbols before the arrow sign as negative and proceeds to find the earliest logical connective.[229]

This was a memory-demanding effort (since each new, simplified proposition generated by the rules would also require seventy-two available memory address), but the IBM 704 was still able to prove all two-hundred-twenty theorems from the first three chapters of *Principia* in three minutes. Wang's rules and representations for *Principia*-by-computer introduced yet different new processes and materials than those developed by Newell, Shaw, and Simon. This was a different *reformalism*.

Also interesting is that, like Newell, Shaw, and Simon, Wang made use of a tree diagram to represent what proofs looked like in the System P. It is very common, in fact, throughout computer science to represent "searches" of different kinds with a tree structure. However, what the nodes and branches of those trees represent can be very different and can have built into them different kinds of processes, assumptions, motivations. Here is one case where an apparent similarity opens in to high level

[229]Wang, "Toward Mechanical Mathematics," p. 6.

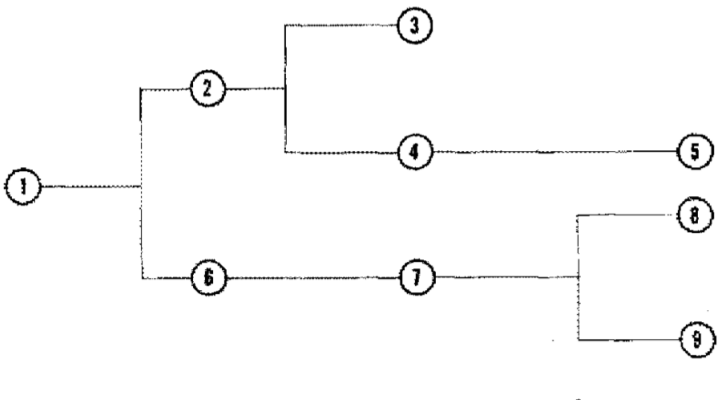


Figure 2.1: Tree Diagram of Program P Proof. Wang, "Toward Mechanical Mathematics," p. 6.

and low level differences in program design and implementation. Recall that Newell, Shaw, and Simon used a tree-structure to represent the "working backwards" and "sub problem chaining" mechanisms that they believed simulated human heuristics in logical theorem proving. As shown in Figure 2.1, Wang too proposed that with his System P, "each proof is a tree structure."

But in his tree, the "branches" represent different processes and different relationships than did the "branches" of Newell, Shaw, and Simon's trees. Each branch in Newell, Shaw, and Simon's trees represented the process of "subproblem generation" by which, the computer would generate a set of subproblems ("child nodes") for some logical proposition (the "parent node") that, if true, would lead that proposition in one permissible inference step. The branches are supposed to represent a human practice, a human search heuristic, as performed by a computer. In Wang's tree, the child nodes are produced by the iterative application of rules for breaking apart a given logical proposition into increasingly simple formulas.

The two trees are also "growing" towards different outcomes. Both trees represent a trajectory towards "simplification" - but the way in which is proposition is being

simplified is different in each case. The branches in Wang's trees approach "atomic formulae" - the most basic constituent elements of a given proposition. Moreover, Wang's branches guarantee this outcome: every well formed proposition can broken down into atomic formulae in the way that Wang's program proceeds. His trees represent a guaranteed (though sometimes computationally costly) path to a proof. However, Wang's branches *do not* represent a human practice. People would only be able to do this "pulling apart" in simple cases, and would be unlikely to go about actually proving theorems this way. Newell, Shaw, and Simon's trees, conversely, approach the axioms of propositional logic - the goal is to trace a path from a desired theorem to the axioms which, if possible, would provide a proof. However, there is no guarantee that the axioms can be produced in this way. Newell, Shaw, and Simon did not mind not having a guarantee because practicing human mathematicians seldom do. The two "proof trees" represent different processes and goals and speak to the two quite different *reformalisms* that Newell-Shaw-Simon and Wang enacted on *Principia.*

After describing the System P in "Toward Mechanical Mathematics," Wang goes on to speculate how a computer might be used to take on the more complicated theorems from *Principia* - those from the predicate calculus presented in later chapters. Here he leaves behind the rules for breaking apart statements from the propositional calculus (which would not serve to prove theorems in the undecidable predicate calculus in the same way) and imagines how one might go about using Herbrand's Theorem. He imagined four "steps" that would suffice to prove a given theorem from the predicate calculus.[230] The four steps were a kind of naive or preliminary implementation of the theorem. Steps one and two served to simplify the statement by eliminating the quantifiers and converting it into a particular form and step three related to the production of correlating statements from the propositional calculus. And Step four sounded a lot

[230]These steps are elaborated in Wang, "Toward Mechanical Mathematics," pp. 13 - 14.

like the core of Herbrand's Theorem:

> Step IV: Make all possible substitutions on these [propositions] obtaining results S_1, S_2, S_3, et cetera. The original [statement] is a theorem if and only if there is a truth-functional tautology among S_1, $S_1 \vee S_2$, $S_1 \vee S_2 \vee S_3$, et cetera.[231]

However, as I've said a few times in the chapter so far - it wouldn't actually be possible for the computer to "make all possible substitutions" in this way because it would often include even more cases than computers could handle, especially in the 1950s. Wang was imagining the possibility of using Herbrand's theorem, but had yet to devise a way to do so, saying "A program for [this method] has no yet been written. It seems clear that certain auxiliary procedures will be useful in reducing the running time and extending the range of application."[232]

Later, in the summer of 1958, Wang worked out some "auxiliary procedures" that enabled him to implement a program capable of proving all of the theorems in predicate logic from *Principia*. The program was still clearly limited and still faced some of the unavoidable difficulties contained in Herbrand's Theorem - but it was powerful enough to deal with the theorems in *Principia* and make a real case for the implementation of abstract logical procedures. Those "auxiliary methods" are the *reformalism* that I take to be most significant in his work. While proving already known theorems - those from *Principia* - the Program P also revealed a new property of those propositions pointing to the epistemological significance of implementation: first, the "proofs" produced by the Program P were a new kind of proof that a person could not make. Moreover, Wang identified *new knowledge* about predicate logic in the behavior of the Program P. New things became known and in new ways in the design, implementation, and execution of the Program P.

[231] Wang, "Toward Mechanical Mathematics," p. 14.
[232] Wang, "Towards Mechanical Mathematics," p. 15

Reformalism II: Finding Patterns in the Predicate Calculus

Recall that Herbrand's Theorem establishes a connection between the propositional and the predicate calculus: for every statement P from the predicate calculus it is possible to construct a corresponding infinite series of statements $S_1, S_2, S_3, \ldots$ from propositional logic. If P is a theorem, then for some number N the disjunctive series $S_1 \vee S_2 \vee \cdots \vee S_N$ will be a tautology. However, recall also that the catch with Herbrand's Theorem is that, one [the imaginary machine or person implementing the hypothetical method] can never tell when P is not a theorem. The hypothetical agent checking all these disjunctive series could never know whether or not by adding one more S_i he/she/it would discover a tautology or whether no added S_i will ever produce a tautology. Herbrand's Theorem could not be implemented "as is."[233]

Wang decided instead to see if there was a way to use the computer to check for *counterexamples*. His question was this: were there structural properties of $S_1, S_2, \ldots$ such that it would be impossible for $S_1, S_1 \vee S_2, \ldots$ ever to be a tautology? If such

[233]Wang did in fact consider another possibility, which he also rejected because it was unfeasible for actual computation. For some kinds of propositions in the predicate calculus, there was a known method for calculating N. With that number in hand, the computer would know exactly how many statements from the propositional calculus to construct, and then it could check if their disjunction was a tautology or not. The way to compute N, however, was "hardly feasible even on machines" and so Wang rejected this other possibility as well. See Wang, "Proving Theorems by Pattern Recognition," p. 222. It is no longer particularly common in computer science publications for authors to describe their "en route" failures - the ideas that they had and discarded for reasons of elegance, efficiency, or feasibility. However, Wang and also Newell, Shaw, and Simon included many such discarded options even in their published works. In fact, the presentation of possibilities and steps in development that Wang presents in his published works resembles quite closely the way those developments appear to have taken place in his unpublished notebooks. Not all of his notebooks survive from this time, but those that do show similar possibilities being imagined and discarded en route to his actual implementation. It is interesting that some of the behind the scenes processes are present in early published works, and it offers a window to historians and to other practitioners of the time, into the *kind* of thinking and problem solving that went into early program implementation. Wang's notes on this implementation can be found especially in RAC-RU-HW, Series 3, Box 97, Folder 1369- [Notebook, Bell Telephone Laboratories, 1959]; Series 3, Box 37a, Folder - [Unlabeled Notes, 1957 - 1958].

a structural property existed it could be used to *disprove* statements that were not theorems - if that property was present, no tautology would be possible and therefore the statement couldn't be true. And the structural property of $S_1, S_2, \ldots$ that Wang used was given by the method for constructing them, which was part of Herbrand's Theorem. Wang summarized the core to his approach to implementation as follows:

> The writer now feels that a more basic step is to eliminate in advance useless terms among S_1, S_2, etc. or alternative, instead of actually constructing and testing the disjunctions, examine in advance, for each given problem, all the possible course along which counterexamples to $S_1, S_1 \vee S_2$, etc. may be continued. Using the second alternative, we obtain at the same time a disproving procedure for most cases. The detailed techniques for achieving these goals are here called the method of proving theorems (and disproving theorems) by pattern recognition, or, more specifically, the method of sequential tables.[234]

Wang wanted to mobilize computers for the task of recognizing structural possibilities for the statements in propositional logic that corresponded to a given statement in predicate calculus, in order to see whether it was possible for no tautology to exist.

This method - of looking for the possibility or impossibility of counterexamples as a way to avoid the impossible task of traversing an infinite series not knowing when to stop - is used often in mathematics. It is a tool for avoiding infinity. However, the patterns that Wang sent the computer looking for and the methods for recognizing them were beyond the capacity of unaided human use. These were new patterns and new forms of recognition than those involved in mathematical practice prior to the introduction of computers. Worth noting too is that there was a certain element of "brute force" to this implementation. The computer was going to be tasked with checking many cases. However, that does not mean that this was either an obvious or a trivial use of computers, e.g. to simply apply the rules of inference to a set of axioms

[234]Wang, "Proving Theorems by Pattern Recognition," p. 223.

in order to deduce all possible consequences. The program in fact constituted a "short cut" or "trick" for managing what would otherwise be an impossibly infinite set of cases. In fact, Wang made a note to himself in the margins of his notebooks that it would be "Important to find some examples difficult by brute force yet simple by pattern recognition" to demonstrate the comparative merit of his implementation.[235] This was an "intelligent" use of computers, but one that capitalized on the speed and efficiency of computing machinery. This was a method of "buying originality with plodding."

The method of sequential tables, or "pattern recognition" that Wang devised for his computer program was a method of looking for ways that a disjunctive series could *fail* to be a tautology, rather than looking for a tautology. Wang went looking in Herbrand's Theorem for structures or patterns that the computer could exploit. Wang's attention was turned to *different properties* of Herbrand's Theorem. No one was looking for these patterns before, perhaps because there was no reason to. The patterns became relevant for Wang because he was rethinking Herbrand's Theorem with the computer in mind. He was seeking an actionable *implementation* and this diverted his gaze from the kind of question and structure upon which proof theorists generally turned their gaze.

The method he devised works like this: Herbrand's Theorem gives a method for producing a series of statements $S_1, S_2, \ldots$ in the propositional calculus that correspond to a given statement, P in the more complicated predicate calculus. At the most basic level, those propositional statements are constructed as follows: first, P is simplified into a standard form, and the statements S_i are produced by substituting all possible values for the variables in P into the simplified form.[236] The reason the series of S_i is

[235]RAC-RU-HW, Series 3, Box 97, Folder 1369- [Notebook, Bell Telephone Laboratories, 1959], p. 148.

[236]The process of simplification uses known methods for eliminating quantifiers and standardizing logical operators, and so on.

usually infinite is because the variables of P usually have an infinite range of possible values (like the integers, for example).

Each disjunction $S_1 \vee \cdots \vee S_N$ may provide a "certificate" that the original statement was a theorem - if the disjunction is a tautology, then P is a theorem - that is the heart of Herbrand's theorem. Wang realized that since the statements S_i are all *obtained in the same way* - namely by assigning values to variables in P, there are structural similarities between them. Crucially, sometimes multiple statements S_i will contain the same variables in such a way that guarantees no tautology will be found.

The method of sequential tables attempts to find a pattern in the S_i that guarantees there is a truth assignment to the variables in the S_i that simultaneously falsifies all of them. Due to the structural similarity in the statements (and that they are produced out of a statement in the predicate calculus that has *finite* length), Wang shows that there are only a finite number of "patterns" in the variable assignment that could falsify each S_i. And since the statements *share variables*, applying one pattern that falsifies statement S_i may make it impossible to apply another pattern to falsify S_j because some of the variables in S_j have already been assigned a truth value.

This "pattern recognition" method either demonstrates that no pattern can falsify all the S_i simultaneously and therefore there must be a tautology; or, it finds a pattern that could be used to construct a variable assignment that simultaneously falsifies all the statements S_1, S_2, etc. and there must be no tautology. Because there are only a finite number of patterns at play, this method will also terminate for any statement in the predicate calculus.

Although the method would always terminate, it would still usually take an enormous number of steps. For very simple examples it would be possible for a person to check these patterns and value assignments, but the size of the table (each of whose rows is a particular variable assignment) grows *exponentially* with the appearance

of variables in S_i. These patterns weren't ones that people could recognize and the method of recognizing them was beyond reach of unaided human combinatorial ability: "the method of sequential tables, seems to be a new feature that goes beyond the general method of pattern cognition."[237] These, I claim, introduced new meanings to the words "pattern" and "recognition" - meanings that were not predicated on human visual perception or with inscriptions on the page. These should perhaps be called "algorithmic patterns" since human access to them is indirect, once removed - people could survey the operations that produced and analyzed the patters, but not the patterns themselves.

Nonetheless, or perhaps even *because* access to these patterns had to be primarily indirect, Wang elected to describe program function on sufficiently simple examples that the steps of the method could be surveyed by a human reader. Even these examples left out some information, but they were meant to be illustrative rather than comprehensive. His first example is the following: *Example (1):* $(\exists x)(y)(z)[(Gyy \wedge Gxx) \implies (Gzx \wedge Gzz)]$.[238] In this formulation, G is what is called a "predicate function." Each G takes some variables (here, G is a binary function taking only 2), and it tells us whether or not a given relationship holds between them. A possible G could be, for example, Gab = *both a and b are even* or Gab = *a is the sum of b and its square.* Each instance of G can have a value of true or false depending on whether or not the relationship captured by the specific predicate function holds between a given pair of variables (that can be assigned domains of numbers). The predicate calculus is set up in such a way that practitioners need not give a concrete interpretation of G in order to study the formal relationships that interest them. That is to say that if a statement

[237]Wang, "Proving Theorems by Pattern Recognition," p. 223.

[238]Wang, "Proving Theorems by Pattern Recognition," p. 222. Wang uses the symbol & rather than $\wedge$ for the logical operation "AND" and the symbol $\supset$ rather than $\implies$ for "IMPLIES" but I have altered the symbols to remain consist with my exposition elsewhere in the chapter. He intends the same logical relations by these symbols.

is a theorem, it will hold for all possible well formed definitions of G.[239] The example statement can be read as follows: "There exists a variable x such that, for all variables y and z,[240] if a given binary relationship obtains between y and itself, and that relationship holds between x and itself, then it follows that that relationship must also hold between some variable z and x and z and itself. It turns out that this statement is a non-theorem, and Wang's method should therefore produce a counterexample.

Herbrand's theorem constructs an infinite series of statements from the propositional logic $S_1, S_2, \ldots$ such that (1) is a theorem if, and only if, there is a number N such that the disjunction $S_1 \vee S_2 \vee \cdots \vee S_N$ is a tautology. Each predicate function G has a value of either true or false depending on the value assigned to its arguments. For a fixed set of arguments, the value of G is a variable in the propositional calculus. The variables that appear in S_i are exactly these. So in the example case, each S_i is produced by assigning a certain value to x, y, and z here treated as integers. For Example (1), the first four propositional statements constructed according to Herbrand's theorem are:

$$
\begin{aligned}
&S_1 = (x, y, z) = (1, 2, 3) : (G_{22} \wedge G_{11}) \implies (G_{31} \wedge G_{33}) \\
&S_2 = (x, y, z) = (2, 4, 5) : (G_{44} \wedge G_{22}) \implies (G_{52} \wedge G_{55}) \\
&S_3 = (x, y, z) = (3, 6, 7) : (G_{33} \wedge G_{66}) \implies (G_{73} \wedge G_{77}) \\
&S_4 = (x, y, z) = (4, 8, 9) : (G_{44} \wedge G_{88}) \implies (G_{94} \wedge G_{99}) \\
&\ldots [241]
\end{aligned}
$$

These are the first four statements generated by Herbrand's theorem. If these, or

[239] These predicates can also describe relations and take variables from nonnumeric branches of mathematics, for example if a and b are sets, rather than numbers Gab could mean that a and b contain no shared elements. The freedom of G is born out of the desire to enable logicians to study the formal structure of mathematics independent of its content. I will talk about the variables of G as being the integers.

[240] In later formulations of logical symbolism, the $\forall$ symbol is used to denote universal quantification, or "for all." However, in *Principia* and in Hilbert-Ackermann, universal quantification is denoted simply by stand-alone parentheses rather than an explicit symbol.

[241] Wang, "Proving Theorems by Pattern Recognition I," p. 222.

any other prefix can be shown to constitute a disjunctive tautology, then (1) is a theorem. If (1) is not a theorem, then such a disjunctive tautology will be *impossible* among all prefixes of S_i. Rather than trying to find disjunctions in the statements $S_1, S_2, \ldots$, Wang's method attempts to prove that no possible truth assignments to the propositional variables can make every S_i false. Intuitively, what Wang's method relies on is the possibility of fixing the values of the propositional variables (e.g. G_{xx}, G_{yy}, G_{zx}, G_{zz}) so that if a disjunctive tautology is impossible, this will be forced in to view in a finite number of steps.

It turns out that (1) is *not* a theorem, and the program reveals it as such in the following way. Recall that each G can have one of two values - true or false - depending on whether some relation holds between it's two input variables upon a given value assignment. For the statement (1) each S_i contains four instances of G: Gxx, Gyy, Gzx, and Gzz can have either the value of true or false. A comprehensive truth table for S_i would therefore have $2^4 = 16$ rows . However, the $\implies$ "implies" logical relation is only false if the left hand side is true and the right hand side is false. In each expression S_i the left hand side will be true only when Gxx and Gyy are true. The right hand side will be false if either or both of Gzx and Gzz are false.

So the subset of the truth table for which $(Gyy \wedge Gxx) \implies (Gzx \wedge Gzz)$ is false consists of only three rows:

Gxx	Gyy	Gzx	Gzz
t	t	f	t
t	t	t	f
t	t	f	f

The program's first substantial operation is to create this kind of truth table, enumerating the possible assignments of true and false to every appearance of G in an

expression that render the expression as a whole false. Each row of the table provides one "pattern" by which the statements S_i may be made false. If it is possible to choose one row of the table to falsify each of the statements S_i such that the chosen rows do not contradict each other (i.e., by requiring that some variable be both true and false), then the disjunction of any prefix will not be a tautology because this constitutes a falsifying assignment. Conversely, if it is not possible to choose rows that falsify each of the S_i, then there must be a tautology and the original statement is a theorem.

In this example, $S_1 = (x, y, z) = (1, 2, 3) : (G_{22} \wedge G_{11}) \implies (G_{31} \wedge G_{33})$. According to the *second* row of the table, we can make S_1 false by making G_{11} true, G_{22} true, G_{31} true, and G_{33} false. However, with this assignment of truth values, we *force* S_3 to be true because there, G_{33} appears on the right hand side of the equation, rather than the left: $S_3 = (x, y, z) = (3, 6, 7) : (G_{33} \wedge G_{66}) \implies (G_{73} \wedge G_{77})$. Indeed, in this example *every value of z will eventually be assigned to a variable on the left hand side of some* S_i *rendering that statement true.* As such, the second row of the above table is not a candidate for falsifying all S_i in this example. The same holds for the last row of the truth table in which G_{zz} is also falsified.

In this way, the program checks each way of making S_j false to see if this requires that another S_k be true, according to the construction of each S_i by Herbrand's theorem. Wang shows that, as it turns out, for all statements in *Principia,* these patterns can be detected mechanically. As will be discussed in the next section, this was a surprising result yielded by working with the Program *P*. This illustrative example obscures the complexity of most such problems. In general, there may be more than one predicate and each predicate may have any parity (unary, binary, etc). As a consequence, there will typically be many more patterns of possible falsification (false rows in the truth table).

In Example (1), the first row in which Gxx is true, Gyy is true, Gzx is false, and

Gzz is true is the only candidate for failure. In the case of S_1 from this example, this model of failure would make G_{11} true, G_{22} true, G_{31} false, and G_{33} true. Crucially, the false term G_{31} never appears on the left hand side of any S_i because its arguments are not equal and therefore it has the wrong *form*. Moreover, the S_i constructed by Herbrand's theorem will never assign the same value to x and z, so using row 1 of the table will never force a false variable to appear on the left.

Here I have not elaborated the discrete operations that the program actually follows (indeed, Wang gives a more in depth but still incomplete survey of the operations for two equally simple examples that occupy some 15 pages)[242] but I have walked along the kind of steps and case checks that the program performs. These steps can be mechanized for the type of statements appearing in *Principia*.

First the program constructs a table, each row of which is a possible way that G (and any other predicates) could be made false. Wang shows that when the statement has a particular form, which it turns out that all propositions in *Principia* do, then there are simple rules that can be used to eliminate rows of the table as candidates for falsifying the S_i. He also shows that, in these cases, a given statement is a theorem if and only if all rows are eliminated by these simple rules. The program then repeatedly applies these rules to eliminate rows from the table until either the table is empty, or a tautology is forced. The method of sequential tables operates on the finite table of possible falsifying assignments, which circumvents dealing directly with the infinite sequence of propositional statements. This insight and the tools with which Wang actualizes it constitute a rethinking and retooling of Herbrand's theorem.

In the previous chapter describing the Logic Theory Machine, I was focused on a very low level of implementation - the assignment and manipulation of values in words of computer memory. My description of Wang's program has focused on a somewhat

[242]See Wang, "Proving Theorems by Pattern Recognition II," pp. 9-22

more abstract level. These operations are, however, I claim still part of the *implementation* to be differentiated from abstract algorithm because they were devised in service of the creation of a running program. Moreover, abstractions are required for any tractable discussion of this method because, in this case, the significant transformation of proof was introduced in the method rather than in the details of the existing IBM-704 assembly language into which the method was translated. This method was focused on *efficiency* and the elimination of "useless terms" that rendered Herbrand's Theorem on its own impossible to implement. This differentiated his method from Herbrand's high-level algorithm: "The basic ideas of the general method of pattern recognition, though not the special addition of the method of sequential tables directed at efficiency, go back to Herbrand."[243]

Moreover, Wang's method differentiated in this regard from the one previous attempt of Martin Davis and Hilary Putnam to create a Herbrand-based algorithm. They too did not sufficiently accommodate the finiteness and practical limitations of actual computers: "their method is concerned only with the last stage, viz., that of testing each disjunction, [so] it can of course do nothing with nontheorems. [...] Moreover, since it provides no decide for deleting useless terms among S_1, S_2, etc. it is not likely to be of use even when a formula is indeed a theorem."[244] That is to say that the Davis-Putnam method actually checks every disjunctive prefix for a tautology, which may never terminate. So although this description of Wang's method doesn't get down to the level of bits of memory as in the previous case, it is still "less abstract" than Herbrand's theorem in the sense that it was designed with an actual limited computing machine in mind.

To say that Wang's method is an "application" of Herbrand's theorem is true but

[243]Wang, "Proving Theorems by Pattern Recognition I," p. 223.
[244]Wang, "Proving Theorems by Pattern Recognition I," p. 223.

over-simple. The development of his method consists was an act of translation through the lens of a *specific* machine and a *specific* ideology - one in which Wang sought to make an abstract idea *useful* and in so doing, surpass the capabilities of unaided human cognition. The method bears traces of both elements of the ideology. He clearly begins with Herbrand's theorem and its related results, and looks for sites where it becomes practically impossible to compute. In those moments, he exploits parts of the theorem to devise previously unhelpful tricks (unhelpful because still unusable by people) and forges his own insights. There is no automation without invention.

The way that Wang wrote and diagrammed about this method in his notes is indicative of the method's incompatibility with human use and with paper media. Very few examples of actual tables with their rows of variable assignment filled in can be found in Wang's publications or his notes. Instead, his notes contain *unfinished* tables, the *outlines* of table, or *sketches* of tables. These gesture to the kind of thing the computer is doing, but do not *actually* do it. The first time the phrase "method of sequential tables" appears in his notes, shown in Figure 2.2.[245] There, Wang was exploring the kinds of "patterns" that corresponded to a particular predicate statement, but he leaves us with an empty table bordered by numbers like $2^{(k+1)^2N}$ and $2^{(k+1)N}$ that indicate the number of cases involved - clearly out of human reach.

Where he writes "if and only if every one has a repetition," Wang refers to the recurrence of variables in various S_i that makes the method work. But most important about this table, to my mind, is that it is *empty*. The *actual* tables, with all the rows populated with values and surveyed in search of certificates that some statement is a theorem or not only ever existed in the storage systems of the IBM 704 computer upon which Wang ran the program in the summer of 1959.

[245]The G_i on the vertical axis are the variables in the various S_i, which are in turn generated by the assignment of iterative values to the variables of the original predicate statement.

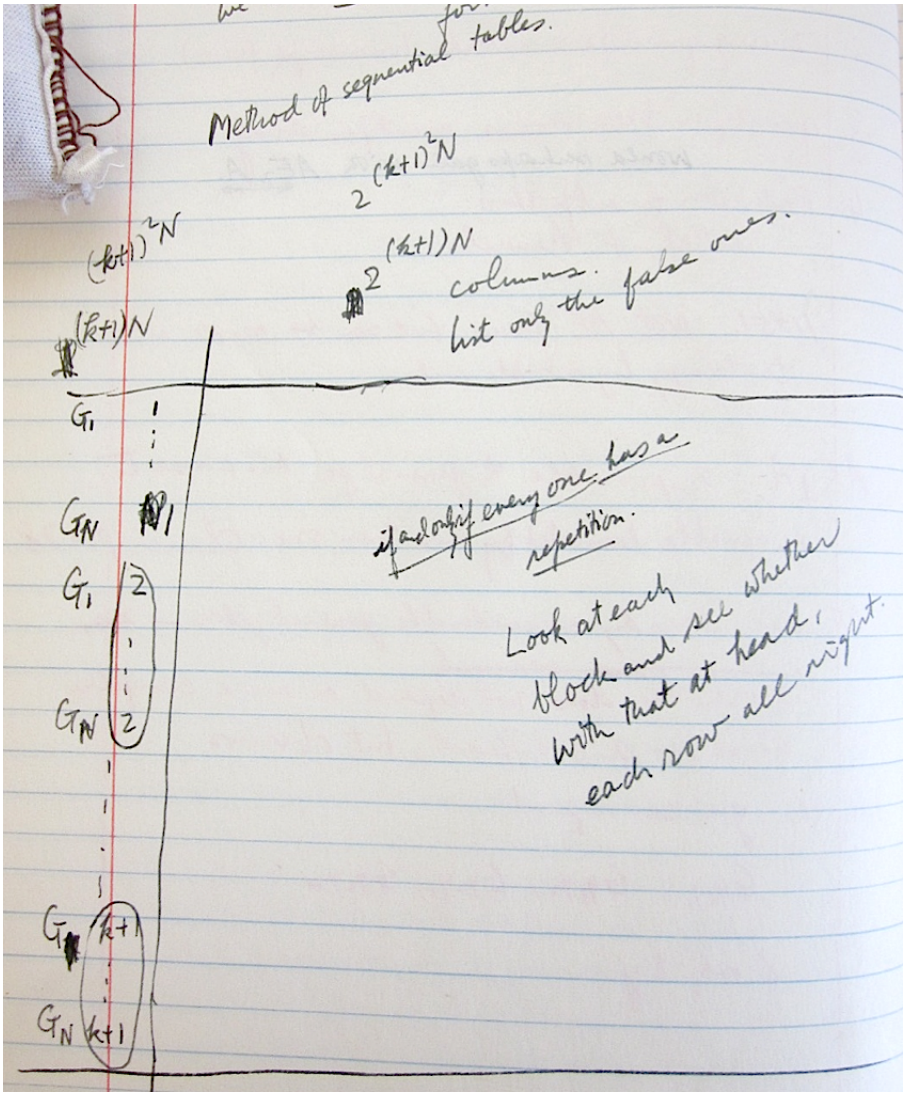


Figure 2.2: Hand Drawn Pattern Recognition Table from Wang's Notes. RAC-RU-HW, Series 3, Box 97, Folder 1369 - [Notebook, Bell Telephone Laboratories, 1959], p. 134.

This is where another set of new materialities enter into the implementation. Wang actually wrote and ran the executable program and this too was part of what it meant to put Herbrand's Theorem to work on a computer, part of this *reformalism*. Whereas Newell, Shaw, and Simon created their own programming language in order to implement the Logic Theory Machine, Wang made use of an existing one - the SHARE Assembly Programming language or SAP, though with the added "impurity" of Bell's Labs unique subroutines for reading and writing on memory tape.[246] The act of transforming his method of pattern recognition and translating it into the obtuse formality of a programming language proved an enlightening task for Wang - one in which he encountered different perspectives about mathematical practice. He remarked in his notebook that "[p]rogramming is an interesting strange intellectual experience. It leads one to conclude that nearly all math contains errors. Surprising they work - a different criterion of mistakes too."[247] Human mathematicians adhered to certain standards of rigor, success, and failure. The computer was held to different ones, but these latter were introduced to mathematics in the work of mathematical programming.

As with the System P described in the previous section, the Program P accepted as input a single punched card whose seventy-two columns contained a seventy-two symbol representation of a single statement from the predicate calculus contained in

[246]SHARE was a community of IBM 704 and IBM 709 computer practitioners who sought to achieve some standardization in programming languages. Newell, Shaw, and Simon faced a particular challenge in working with the JOHNNIAC because that computer had only ever been understood as a numerical processing machine and its existing programming tools were oriented towards the coding of numerical tasks. The IBM 704, however, was understood as a more versatile machine already in its inception. SAP was a language designed to be flexible for the implementation of numerical and nonnumerical tasks, and Wang was happy to work within it. For a historical discussion of the SAP language see Mark Priestly, *A Science of Operations: Machines, Logic and the Invention of Programming* (Berlin: Springer, 2011): 205 - 209. As for the Bell Labs sub routines, these had been designed to accommodate particular Bell Labs equipment and Wang notes that by altering a few lines of code, the program could be made "portable" to other machines. See Wang, "Proving Theorems by Pattern Recognition," p. 225.

[247]RAC-RU-HW, Series 3, Box 97: Folder 1369- [Notebook, Bell Telephone Laboratories, 1959], p. 106. By "they" I imagine he means the results of less formalized and less formal mathematical work.

Principia. The program itself was given to the IBM 704 by way of three-thousand, two-hundred punched cards of instructions and operations that occupied thirteen-thousand words of core memory during program runs. Wang was eager to emphasize that the program was not that resource-consuming, indicating that with some editing, it would be possible to "fit everything into a machine with 8000 words. Auxiliary storages are not needed except that, as a convenience, tapes are used to avoid going through on-line input-output equipments."[248] Wang's abstract algorithm for pattern recognition - already a *reformalism* in the sense that it involved new formal and material tools for *Principia* and new computer-inflected perspectives on *Principia* - was also translated into the languages and materials of the IBM 704 in which the *actual* pattern recognition took place.

Here again, someone crafted a set of tools that were *not amenable to human use*. And these tools were added to the repertoire of theorem-proving materials and practices and ideas where none like it were before. The Program P produced proofs with a different form than those of Whitehead and Russell and those of the Logic Theory Machine. This form of proof would either be a demonstration that it was *impossible* to produce a tautology in the disjunction of S_i or that it was impossible not to produce one. Although the steps would not be surveyable by a person, that output "certification" was. In proving all of the more than three-hundred fifty theorems from the first nine chapters of *Principia*, the Program P output one-hundred ten pages with sixty lines each presenting these certifications whose status as "proofs" was grounded in the proof theoretic results out of which the method of pattern recognition was fashioned. Here was a form of proof that had a new relationship to existing results - rather than "calling on" Herbrand's Theorem as, for example, a step in a demonstration, these proofs were fashioned out of the pieces of Herbrand's Theorem as Wang transformed them into

[248] Wang, "Proving Theorems with Pattern Recognition," p. 225.

lines of code. This is how Wang believed we should proceed with computers: "Man will have to devise methods or methods for devising methods, the machine will use the methods to do things which man cannot feasibly do."[249] Wang both *rethought* Herbrand's Theorem through the lens of what computers could do, and then *actually transformed it* into things that computers could do by devising a new process and implementing it with technologies of digital computing. And Wang, like other theorem-proving practitioners, believed that by proceeding this way, the computer would show us things we didn't know before. Even though Wang fashioned all of the tools that were put to work in the Program P, the Program P produced a result that he did not see coming.

Conclusion: "A Rather Surprising Discovery"

Even if all the Program P had done was prove the theorems from *Principia*, something interesting would have happened. A *reformalism* would have occurred. A logical abstraction would have been translated into an actionable tool kit. The status and meaning and interest of Herbrand's Theorem would have been changed. A new form of proof with a new relationship to existing ideas would have been fashioned. But something even more happened, lending added credence to my claim that implementation has *epistemological* significance for the history of mathematics.

It turned out that the Program P also yielded a *new result*. New knowledge about *Principia* was made visible to Wang in the behavior of the program. One way to describe the result is this: "the theorems in *Principia* are far easier to prove than expected."[250] Within the predicate calculus, there are different "subdomains" of statements that have particular forms, usually given by the combination of quantifiers they

[249]Wang, "Proving Theorems by Pattern Recognition," p. 224.
[250]Wang, "Proving Theorems by Pattern Recognition," p. 226.

contain. For example, some statements are in the so-called $\forall\exists$ class, where $\forall$ is the "for all" quantifier and $\exists$ is the "exists" quantifier. Statements within this subdomain all begin with those one $\forall$ quantifier followed by zero or more $\exists$ quantifiers. For example, $\forall x \exists y : (x + y) > 10$, which reads "for all x there exists some y such that the sum of x and y is greater than 10. Other subdomains were carved out by other combinations of quantifiers like $\forall\exists\forall$. For a statement to be a member of any such subdomain it has to be possible to transform it into an equivalent statement with these particular quantified forms. Logicians in the twentieth century were interested in studying these different subdomains - what are their formal properties? Are some of them harder or easier to prove than others? Are some subdomains of undecidable domains decidable? And so on.

In running the Program P and inspecting the output proofs, Wang discovered that in fact, every predicate calculus theorem in *Principia* falls into one particular subdomain: the $\forall\exists$ domain. Included in *Principia* was a method for simplifying predicate statements, and after applying this method to each statement in *Principia*, they all were in the $\forall\exists$ domain. Although the predicate calculus is an undecidable domain (which is where Herbrand's Theorem is a powerful tool), the $\forall\exists$ domain is *decidable* - one can formulate a procedure (i.e., Wang's method of sequential tables) that would return either "yes" or "no" when given any statement of that form. This was an actionable method rather than an abstract one. It hadn't been shown before that *Principia* contained only theorems of this form. Wang proposed that this was a rather "surprising discovery, which tends to indicate our general ignorance of the extensive range of decidable subdomains."[251] The negative result of the "decision problem" and Kurt Gödel's incompleteness results of the 1930s certainly were a blow to some goals of mathematical logic. However, relative to his computer-assisted discovery Wang wrote

[251] Wang, "Toward Mechanical Mathematics," p. 4.

that "[o]ne may even claim that a new life is given to the Hilbert program of the *Entscheidungsproblem*," by opening up new directions for the study of formal systems.

Just because Wang devised and implemented the method of pattern recognition, the Program P surprised him. In the minds of most computer practitioners, computers have significant potential for producing surprises and novelty in spite of the fact that they can only do what they are programmed to do. But the *kinds* of surprise and the kinds of novelty that different practitioners are after can vary. As discussed in the concluding section of the previous chapter, the Logic Theory Machine was championed for producing a previously unknown and "more elegant" human-style proof of a proposition from *Principia*. Wang was excited about his program for demonstrating to him that all of the canonical theorems from that same text in fact share a fundamental structural property. Computing made possible new perspectives and new results but different perspectives and different results emerged depending on what practitioners were hoping to do with their machines. From Wang's perspective, "[w]ith machines, large masses of well-organized minute details seem to be the only sure way to make the correct surprises emerge."[252] He took tools from the abstract realms of logic and transformed them into tools for managing masses of detail in pursuit of a bridge between abstraction, action, and mathematical practice. In the next chapter we will see yet another approach to automated theorem-proving, yet another example of *reformalism* in which different elements of mathematics were transformed differently, and in which a different kind of surprise was celebrated in the results.

[252]Wang, "Proving Theorems by Pattern Recognition," p. 234.

Chapter 3

A New Collaborator:

Implementing Intuition and Inference

Introduction: Divisions of Labor

Hao Wang was the recipient of the American Mathematical Society "Milestone Award in Automated Theorem-Proving" at a conference in 1982 for his work on the Program P. A second award was given at that same conference - the "Current Research Award in Automated Theorem-Proving." It was given to two practitioners based at the Argonne National Laboratory, Lawrence Wos and Steven Winker. In their acceptance speech, Wos and Winker said the following:

> With the advent of the computer, interest was expressed in attempting to automate the activity of proving theorems. Perhaps understandably, some originally thought that the entire activity eventually could be automated - thought that a computer program could be written to accept a purported theorem, and return a proof. [...] As history proved, this all-powerful program would not be found. Proving theorems in mathematics and in logic is too complex a task for total automation for it requires insight, deep thought, and much knowledge and experience.[253]

[253]Larry Wos, Steven Winker, "Open Questions Solved with the Assistance of AURA," in *Automated*

For all their differences, Wang and Newell-Shaw Simon shared a belief that the computer could be programmed to prove theorems, *on its own*. Both of their programs were designed to take a logical statement from *Principia Mathematica* and to produce a proof of it without any human intervention (beyond the programming). Both imagined the possibility of developing programs that would eventually prove theorems from more complicated branches of mathematics also on their own. Wos and Winker and the automated theorem-proving team they worked with at Argonne believed otherwise: they believed that computers would never be able to prove significant theorems from more complex branches of mathematics on their own.[254]

Instead, the Argonne team wanted to develop *collaborative* theorem-proving programs that would work with a human user to prove theorems. Their approach in some ways synthesized the perspectives of Newell-Shaw-Simon and Wang. They agreed with Newell, Shaw, and Simon that human intuitions, insights, and ideas were and should remain a central element of theorem-proving practices. But unlike Newell, Shaw, and Simon they did not think that those human intuitions could be automated. Newell and Simon sought to identify in human theorem-proving practice and then *implement* them as rule-bound algorithmic computer operations, like "subproblem chaining" and "working backwards." Wos and his team believed that the human insights that underpinned proof search were not the result of any rule-bound process and so could never

Theorem Proving: After 25 Years [*Contemporary Mathematics*, Vol. 29] (Providence, RI: American Mathematical Society, 1983): 73 - 88, p. 73, p. 74.

[254]Some members of the team, among them Steve Winker and Ross Overbeek, were actually employed at the Northern Illinois University until the early 1980s. According to Overbeek, a quarrel with the University Administration regarding the trajectory of their then rapidly growing Computer Science department, provoked his (and many others') departure. Until that time, the Northern ATP group was quite separate from and even in competition with the Argonne group. However after the exodus from Northern, many transferred to Argonne, essentially merging the two teams. In 1983, when Overbeek officially arrived at Argonne, AURA was already under construction and included design features from the Northern group and the Argonne group.

be automated.[255]

Wos and his team agreed with Wang that human reasoning and computation were qualitatively different processes. The latter was good at processing large masses of details and at following inferential paths that would be impossible for humans to follow. They too thought that this feature of computing machines should be exploited, that computers offered a way to surpass "the traditional limitation on the complexity of inference" and exploration in mathematics.[256] However, they diverged from Wang in that they believed that the speed and efficiency of computers without continuous guidance from people would not be able to solve mathematical problems more difficult than those found in the pages of *Principia Mathematica*. They believed that the speed and power of computer should be harnessed for the work of theorem-proving but that it should be directed, in "real time" during particular searches for a proof, by a human user. The Argonne team eschewed the project of total automation and instead aimed to implement what they called a *reasoning assistant* that could collaborate with human users. The users would provide their insights and intuitions about a proof search and the computer program would follow those insights along paths that the people might not be able to pursue.

Another difference was that the Argonne team wasn't interested in *Principia Mathematica* at all. They didn't want to harness a computer to produce proofs of statements *already known to be theorems*. They wanted to see whether or not they could get computers to be useful in more complicated branches of mathematics, i.e. not the predicate calculus: they wanted to work on "problems that were previously unsolvable

[255]Peter Galison documents a similar position held by Luis Alvarez, a prominent 20th century experimental physicist, in "FORTRAN and Human Nature" from *Image and Logic; A Material Culture of Microphysics* (Chicago: University of Chicago Press, 1997) at pp. 403 - 411. Alvarez reportedly believed that humans had the unique ability to "unravel pictures" that could not be "built into a computer" (p. 406).

[256]John Alan Robinson, "A Machine-Oriented Logic Based on the Resolution Principle" in the *Journal of the Association for Computing Machinery*, Vol. 12, No. 1 (January 1965): 23 - 41, p. 24.

by automated systems. Included would be problems considered difficult because of the complex nature of the axiom set, such as set theory or certain database problems, and problems considered difficult because of the length of the proof or the nature of the inferences require in the proof."[257] And they wanted to see whether or not they could get computers to be useful in solving *open problems* - proving or disproving conjectures that weren't known yet to be true or false.

I have traced that new ways of knowing and thinking about logic were produced in the development of both the Logic Theory Machine and the Program P. But here, new knowledge was produced in the more traditional sense: in designing, implementing, and then collaborating with their theorem proving program, the Argonne team purported to have produced proofs of previously unknown theorems. Wos and Winker claimed that "...researchers involved in the effort knew and know almost nothing of the fields from which the questions were selected. Rather than indicating the triviality of the questions (some of which are far from easy to answer), this fact shows the potential value of having access to an automated assistant – a colleague in the form of a theorem-proving program".[258] They argued that the program was not a mere extension or amplification of their existing knowledge of mathematics, but that by collaborating with it, they were able to prove theorems in new ways, beyond the limitations of their existing mathematical knowledge.

This chapter explores the earliest and one of the most successful collaborative automated theorem-proving programs developed by the Argonne team - the Automated Reasoning Assistant (AURA) - implemented and employed during the late 1970s and

[257]Robert Veroff, "Canonicalization and Demodulation" Argonne National Laboratory Technical Report ANL-81-6 (February 1981), p. 6.

[258]Larry Wos, Steve Winker, "Open Questions Solved with the Assistance of AURA" in *Contemporary Mathematics*, Vol. 29. p. 74.

early 1980s, but incorporating an earlier result from 1965.[259] It was designed according to a specific understanding of proof. For the Argonne team, proof consisted of two kinds of processes: intuition, which they reserved for people, and inference, which they wanted to assign to computers. Mathematical exploration, in their view, was guided by the unautomatable insights and intuitions of human beings who had experience and instincts with mathematical knowledge that computers could never achieve. But computers were more powerful at executing complex logical operations and so could be used to "infer" faster, farther, and in new directions than there human counterparts.

In this chapter I explore two cases of *reformalism*: I explore how both intuition and inference were transformed in the design and implementation of the AURA program. In the first instance, even though the Argonne team reserved intuition for people, believing that it could not be automated, they *still* fashioned new forms for it to take. In order for human users to impart their insights and intuitions to a computer, they had to first translate them so that the computer could understand and use them - human intuition had to be made into *input*. In designing the human-user interface for the AURA, one member of the Argonne team - Ross Overbeek - designed what was

[259]My study of the AURA is primarily based on technical reports documenting its design, and the "reference manuals" that were circulated at Argonne. In particular, Brian Smith, *Reference Manual for the Environmental Theorem Prover: An Incarnation of AURA (Automated Reasoning Assistant)*, Argonne National Laboratory, Technical ReportANL-88-2 (March 1988) and Robert Veroff, "Canonicalization and Demodulation" Argonne National Laboratory Technical Report ANL-81-6 (February 1981). Although the former was published somewhat late in the development of AURA, it was primarily a collection of previously existing documents for the purpose of enabling people around Argonne to make use of the system. The central published works describing the system and its results are: Steven Winker, "Generation and Verification of Finite Models and Counter-examples Using an Automated Theorem Prover Answering Two Open Questions" in *Journal of the Association for Computing Machinery* Vol. 29 (1982): 273 - 284; Winker, Lawrence Wos, "Automated Generation of Models and Counterexamples and its Application to Open Questions in Ternary Boolean Algebra," in *Proceedings of the Eighth International Symposium on Multiple-Valued Logic*, (Rosemont, IL: IEEE and ACM Publications, 1978): 251 - 256; Winker, Wos, Ewing Luks, "Semigroups, antiautomorphisms, and Involutions: A Computer Solution to an Open Problem, I" in *Mathematics of Computation*, Vol. 37 (1981): 533 - 545; Wos, "Solving Open Questions with an Automated Theorem-Proving Program" at the 6th *Conference on Automated Deduction*, [*Lecture Notes in Computer Science* Vol. 138], ed. Donald Loveland (Berlin: Springer-Verlag, 1982): 1 - 31; Wos, Winker, Lusk, "An automated reasoning system," in *AFIPS Conference Proceedings*, Vol. 50 *1981): 697 - 702.

called a "weighting mechanism" by which human intuitions were input to the computer as quantitative measures of the relative significance of certain inference paths. These "weighting mechanisms" were quite different from the so-called *eureka* moments with which Larry Wos identified unautomatable human insights. This transformations in "intuition" is the subject of Section 3 "Reformalism I: Inputting *Eureka*."

The second case of reformalism concerns logical inference - the part of proof that the Argonne team wanted to offload onto the computer. And, in contrast to intuition, they wanted to make a big change. During the mid-1960s, a Classicist turned computing practitioner by the name of John Alan Robinson was visiting Argonne. He was interested in how computers might make possible entirely new forms of inference in mathematics. Inference rules are the principles that enable mathematicians to get from one idea to the next, they are the justification for why something follows from another. What counts as a sound inference, an acceptable step, has changed through history but Robinson suggested that always *human psychology* was at the core of those rules. Inference rules were designed "to be apprehended as correct by a human being in a single intellectual act."[260] He wanted to design inference principles that might not be apprehensible to the human mind as correct in a "single intellectual act" but that rather capitalized on the kinds of acts computers could perform. In 1965, Robinson designed such an inference principle - called Resolution. The Resolution principle and its many variations, were and are still built in to many theorem-proving programs. And it was built in to the AURA - Resolution constituted the computer's contribution to the human-machine collaboration. The Resolution Principle is the subject of Section 4 "Reformalism II: "A Single Step.""

But more, the AURA was full of surprises. Although the program could only do precisely what it was programmed to, and although users would impart their hunches

[260]Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," p. 23.

about how to proceed, they could not know in advance what the consequences of their instructions would be. Indeed as I have often stated throughout the dissertation, if they could, they would have had little need for a fast computer to do this work for them. Especially when the computer was employing a rule like Resolution that exceeded easy human cognizing, the computer would end up in surprising places. As such, Argonne researchers studied printouts of thousands of clauses from run after run on AURA in order to understand what was going on when the engine looked for proofs. To accommodate their *a posteriori* revelations, the Argonne group adopted a committed experimental paradigm for automated theorem-proving, in which AURA was constantly improved and redesigned. I propose that in fact, those prized human insights and intuitions that the Argonne researchers reserved for the human, rather than materializing from the cognitive ether, began to emerge from their intimate and prolonged experimental work with the computer program. Experimental experience determined the character, form, and relevance of their input intuitions for AURA proof searches. By privileging and isolating a traditional notion of human thought in their design, the Argonne team in fact made possible radically new forms of intuition and insight grounded in knowledge of computational behavior. New experiences and practices - namely working with the AURA - shaped new kinds of intuitions than those the team had in mind before. Even though they cordoned off intuition as something uniquely human, something impossible for computers, they still transformed it through encounters with computing. Intuition was made into computer input and intuition was derived from new kinds of experiences with computers. These *practices* and *experiences* that constituted theorem-proving with AURA are the subject of Section 5 "The Quickest and Surest Way to Insight."Before investigating these *reformalisms*, however, this chapter begins with a discussion of Argonne's place in the postwar landscape, and its role in the disciplinary trajectory of automated theorem-proving.

Disciplining Automated Theorem Proving

The architects of the AURA were based, in varying capacities, at the Applied Mathematics Division of the Argonne National Laboratory near Chicago, IL throughout the 1970s and 1980s. The core of the group was constituted by Larry Wos, Steve Winker, Ross Overbeek, John Alan Robinson, Brian Smith, and Ewin "Rusty" Lusk. Argonne was a state organization, funded by the Department of Energy. It was founded during the Second World War in conjunction with the University of Chicago Fermilab. Fermilab was affiliated with the Manhattan project and, under the direction of Enrico Fermi, was tasked with the generation of a self-sustaining nuclear chain reaction. After the war, Argonne remained operational as a scientific research center after the close of the war. Given that Argonne was established primarily as a nuclear reactor testing laboratory for the Manhattan Project, it might be surprising to discover a team of automated theorem-proving researchers there. Its existence seems to have been the achievement of William Miller - the Director of the Applied Mathematics Division at Argonne from its founding in 1956 until 1965.

Much like the Numerical Analysis Department (later Programming Department) at the RAND Corporation, The Division was originally intended to provide information processing services, computing services, and mathematical analysis to other research groups at Argonne. In spite of its "service" role, Miller was deeply interested in computing, however, believing that its limitations and possibilities should be the subject of inquiry in its own right. Historian Donald MacKenzie also reports that Miller "had a broad conception of his role. He was thinking a great deal about 'what could be automated'... and was 'much influenced' by the codeveloper of the Logic Theory Machine, Allen Newell".[261]

[261] Newell was another ATP researcher. MacKenzie, *Mechanizing Proof; Computing, Risk, and Trust*, Cambridge: MIT Press, 2001, p. 78.

Miller, apparently interested in computing research for its own sake, attended a number of computing conferences in the 1950s at which he encountered Allen Newell, who sparked his interest in theorem-proving as a fruitful domain of computing research and the question of the limits of automation. That Newell was an original motivating factor in the creation of automated theorem-proving research at Argonne is somewhat ironic, given that, as we will see, the team there was central in effecting a separation of automated theorem-proving from Artificial Intelligence in which Newell's efforts were most grounded. Miller managed to allocate resources and personnel for theoretical computer research through the 1950s, the most influential core of which was the automated theorem-proving group, led by his first such hire, Larry Wos. Jack Holl, a historian of Argonne, indicated that Miller "was not modest and believed that his applied mathematics division was engaged in the most important, profound, and far-reaching activity of any group at Argonne".[262]

Wos came to Argonne in 1957 by a somewhat untraditional path. He was a pure mathematician by training, receiving his B.Sc. from the University of Chicago and his PhD from the University of Illinois at Urbana Champagne. It was not uncommon for mathematicians to find themselves at military or industrial research centers in the 1950s, but Wos's reason for ending up at Argonne was somewhat unique. Wos has been blind since his birth. In order to pursue his mathematics education, he and his professors had to develop a braille system with which he could encounter mathematical objects and texts.

It had been his intention to remain in academic mathematics, but he had difficulty securing a position after graduation. He claims that, in spite of offering to share his salary with someone who could do the 'board work' in his classes, and in spite of feeling

[262]Holl, *Argonne National Laboratory, 1946 - 96*, Chicago: University of Illinois Press, 1997, p. 125. Argonne still hosts a highly successful, robust, and exciting team - called the Computational Mathematics group - engaged in enrolling computers in the production of mathematical knowledge.

fully supported by the mathematics department at the University of Illinois at Urbana Champagne, Wos was denied a teaching position by a Dean at the university because of perceived obstacles presented by his blindness.[263] His lack of teaching experience and his encounters with skepticism from his own dean left him feeling nervous about job prospects. He was also newly married and eager to secure a position and an income, so when he saw that Argonne was hiring, he applied.

Wos had never encountered a computer before arriving at Argonne, so it was perhaps somewhat surprising that he would be hired for a position in a computing department. Miller found Wos to be an attractive hire, however, because he was a mathematician. Miller wanted someone who could direct automated theorem-proving research in the Division, and who better than a mathematician to fill that role. Wos apparently had no trouble learning the ropes of computation, claiming that there wasn't a single bug in the first program he wrote at Argonne in 1957.[264] However, it was not until the mid-1960s that Wos turned his efforts in earnest towards the development of a program for automated theorem-proving research. At that time, Wos began arranging for visits from various logicians, engineers, and philosophers. This community would ultimately become one of the most vibrant communities of automated theorem-proving research in history. Wos and the community he fashioned at Argonne would also become instrumental in the disciplinary trajectory of automated theorem-proving.

The early 1980s was a time of disciplinary stabilization for the field of automated theorem-proving. As computing research more generally became increasingly established in institutional, academic, and professional milieus, practitioners of different subfields carved out their intellectual territory and their identity by way of disciplin-

[263]Tim A. Obermiller, "Top of His Game", *University of Chicago Magazine,* 1997, Vol. 89, No. 4 (online edition).
[264]Interview with Wos, November 4, 2010.

ing technologies like journals, conferences, courses, awards, and so on.[265] Automated theorem-proving was among those subfields, grounding and articulating its character through an array of disciplinary tools. I have already mentioned the creation of the "Milestone Award in Automated Theorem-Proving" and the "Current Research Award in Automated Theorem Proving" that were first given in 1982. These awards served to establish criteria of "significance" relative to work in the field.

A number of early 1980s publications also served to "canonize" and consolidate a body of literature whose content would bound the subject. Following the American Mathematical Society Conference at which those awards were given, *Automated Theorem-Proving: After 25 Years* was published in which the proceedings of that conference were collected and framed with actor-historical reflections on the trajectory of the field.[266] In the same year, a two-volume set of "classical papers" from the field was published in 1983.[267] These volumes served to establish boundaries around the kind of work that "counted" as automated theorem-proving and they held up particular contributions - including perhaps especially those of Newell, Simon, Shaw, Wang, and the

[265] Beginning in the late 1960s and accelerating through the 1970s, computer science departments were being established at universities, degree programs were being developed at the undergraduate and graduate levels, and different academic and industrial professions related to computer work were available. It is well beyond the scope of this discussion to offer a comprehensive exploration of the overall "disciplining" of computer science, especially since no account of it has yet been given. However, pieces of the story are told in: Nathan Ensmenger, *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise* (Cambridge, MA: The MIT Press, 2010); Thomas Haigh, "Inventing Information Systems: The Systems Men and the Computer, 1950-1968," in *The Business History Review* Vol. 75, No. 1 (Spring 2001): 15 - 61; Brent Jesiek, "The Origins and Early History of Computer Engineering in the United States" in *IEEE Annals of the History of Computing* Vol. 35, No. 3 (July-September 2013): 6 - 18; David Grier, "The ENIAC, the verb "to program" and the emergence of digital computer" in *IEEE Annals of the History of Computing,* Vol. 18, No. 1 (1996): 51 - 55; William Aspray, "Was Early Entry a Competitive Advantage? US Universities that Entered Computing in the 1940s" in *IEEE Annals of the History of Computing* Vol. 22 No. 3 (July-September 2000): 42 - 87; Michael Mahoney, "Finding a History for Software Engineering" in *IEEE Annals of the History of Computing* Vol. 26, No. 1 (2004): 8 - 19.

[266] In particular Donald Loveland, "Automated Theorem-Proving: a Quarter-Century Review" in *Automated Theorem-Proving: After 25 Years*: 1 - 46.

[267] Jörg Siekmann, Graham Wrightson, eds. *Automation of Reasoning 1: Classical Papers on Computational Logic 1957 - 1966* (Berlin: Springer-Verlag, 1983); Siekmann, Wrightson, *Automation of Reasoning 2: Classical Papers on Computational Logic: 1967 - 1970* (Berlin: Springer-Verlag, 1983).

Argonne team - as significant, and the offered historical reflections of the trajectory of the field.[268] Interesting also is what was excluded from these volumes. For example, *Automated Theorem Proving: After 25 Years* did not contain any articles exhibiting traditional Artificial Intelligence approaches to theorem-proving, like that of Newell and Simon. This was strange given that Artificial Intelligence pioneer John McCarthy presented at the the 1982 American Mathematical Society Conference on which the volume was based, but his paper - "Non-monotonic reasoning and Common Sense Inference" - did not appear in the volume. I was unable to discover why this was the case, but regardless of the reason, the absence of this approach conveyed a particular "bound" around the approaches that were included.

The disciplinary distance emerging between automated theorem-proving and Artificial Intelligence was further reinforced when, also in 1983, Larry Wos of Argonne became the first Editor-in-Chief of the *Journal of Automated Reasoning.* The journal aimed to collect results from automated theorem-proving, logic programming, computational logic and other fields aimed at the provision of "automated assistance for those aspects of problem solving that require reasoning."[269] Uniting these fields was the goal of studying reasoning, deduction, and inference in a computational context without limiting those processes to their various human-based manifestations. Given this difference, it was not surprising that some automated theorem-proving practitioners struggled to publish their work in Artificial Intelligence journals. Another Argonne practitioner, Ross Overbeek recounted the following story:

> When Steve Winker submitted a lovely paper on qualified hyperresolution to one of the main AI journals, a senior editor did not even send it out for refereeing; he just returned a short note stating, "The JACM [*Journal of the Association of Computing Machinery*] is still publishing such pa-

[268]Especially Martin Davis, "The Prehistory and Early History of Automated Deduction" *in Automation of Reasoning 1*: 1 - 28.

[269]Wos, "A Journal is Born" in *Journal of Automated Reasoning*, Vol. 1, No. 1 (1985): 1 - 3, p. 2.

> pers, although I don't know why." This last phrase symbolized the broader AI community in our eyes. Like perceptrons, formal logic had [...] been evaluated and found lacking.[270]

Practitioners at Argonne were central to the "disciplining" of automated theorem-proving, and the slightly more generalized domains of "automated deduction" and "automated reasoning." Even before this flurry of disciplining activity in the early 1980s, they hosted the first Conference on Automated Deduction (CADE) in 1974 - a conference that remains a vibrant annual event for practitioners working in the field.[271]

In 1992, another award for automated theorem-proving was created in conjunction with CADE - the "Herbrand Award," so-called because of the success of programs that, following Wang's, used techniques from Herbrand's Theorem and other related results of proof theory. Larry Wos of Argonne was the first recipient. John Alan Robinson, who consulted at Argonne during the 1960s and who I will discuss in the section on the "Resolution Principle," was the third recipient.[272] Yet another member of the Argonne team, William McCune who architected many descendants of the AURA program was the seventh recipient. Argonne loomed large in the field and its practitioners played a central role in shaping and directing it. Recently at the 2012 CADE, I learned in conversation that a prominent current automated theorem-proving researcher Geoff Sutcliffe had spent some time at Argonne during the 1990s. I remarked that I hadn't known he spent time there to which he responded off-handedly that "oh everyone spent time at Argonne." The Department of Energy cut funding to automated theorem-proving research at Argonne in 2006 resulting in the dissolution of the community that

[270]Ross Overbeek, Ewing Lusk, "Wos and Automated Deduction at ANL: The Ethos" in *Automated Reasoning and its Applications; Essays in Honor of Larry Wos* (Cambrudge, MA: The MIT Press, 1997): 1 - 12, p. 6.

[271]See the Conclusion of this dissertation for a brief discussion of the current state of affairs in the field.

[272]John Alan Robinson also received the AMS Milestone Award for Automated Theorem-Proving in 1985, following Wang.

remained there, but until that time it was central site for the cultivation and circulation of work in the field.[273]

The Argonne community had a different disciplinary orientation than the practitioners discussed in previous chapters. Recall that for Newell and Simon, theorem-proving was not an end in itself but rather just an example of a more general phenomena - symbolic information processing - that they believed could be manifest in computers as in human minds. Their engagement with theorem-proving was limited largely to the 1950s and both moved on to study other problem domains and to the problem of human problem-solving in general. They both became prominent figures in Artificial Intelligence - a quite distinct field aimed at the production of *human-like* reasoning processes in computing machines. People like Wos and Robinson, similar to Wang as discussed in the previous chapter, were interested in processes of inference, deduction, and logical analysis that may bear *no* resemblance to their human counterparts.[274] Also like Wang, Wos - whose perspective was central in shaping automated theorem-proving researchers at Argonne - was a mathematician by training and was motivated by the desire to introduce computers to mathematical problem solving.

However, the Argonne community differed somewhat from Wang as well. Recall that Wang wanted to institute a new field - inferential analysis - *within mathematics*. Automated theorem-proving, however, turned out to stabilize as a field in its own right more closely allied with computer science than with mathematics. Moreover, although the team at Argonne was very interested in attracting the attention of mathemati-

[273]At this time, key researchers still there (like Wos himself) either retired or accepted positions elsewhere: Ross Overbeek to Stephanie Dick, email 6 january 2011.

[274]There were some automated theorem-proving communities that remained committed to the simulation of human methods in theorem-proving, especially a team led by Woody Bledsoe at the University of Texas at Austin. Their work was well respected among more logic-oriented theorem-proving practitioners and was also recognized by certain Artificial Intelligence practitioners as well. However, they were in the minority. See, for example, Woody Bledsoe, "Some Automatic Proofs in Analysis" in *Automated Theorem Proving: After 25 Years*: 89 - 118.

cians or others who might be interested in "conducting research" with a program like AURA, they were also open to extra-mathematical applications for theorem-proving programs.[275] Moreover, as was clear in the design and implementation of the Program P, Wang was working very much within the realm of his expert knowledge - he wanted to apply difficult results from proof theory in order to produce novel insights for proof theory. The Argonne team wanted instead to use computers to newly enable people to work in fields about which practitioners "know almost nothing" by devising a particular division of labor between human users and computers.[276] This represented a different vision for the future of mathematics - in which computers would become like colleagues or assistants for human researchers. The AURA was one of Argonne's earliest programs and it embodied many of their research priorities that would be expanded upon and appropriated throughout the field later and even still today.

Reformalism I: Inputting *Eureka*

The overall design of AURA and the research program in automated theorem-proving at Argonne more generally, was shaped by the perspectives of Larry Wos. Wos advocated the design of collaborative theorem-proving software that would combine what he believed were the very different skill-sets of humans and computers. Human users

[275]Wos, Winker, "Open Questions Solved with the Assistance of AURA," p. 75. In particular, theorem-proving software developed at Argonne was used for the optimization of circuit design, and for the significant field of "program correctness proving." During the 1970s and 1980s, there was growing unease about computers. They were occupying increasingly many and increasingly important positions in the financial, military, political, and industrial infrastructure of the country. And sometimes they didn't behave as they were intended. In order to trust computers, people wanted a guarantee that they would behave as their programmers intended - that programs did not contain errors or bugs that could jeopardize the computer's performance. One approach to solving this problem was to represent computer programs themselves as mathematical systems and then *prove* that they would behave in the desired fashion. Often, and somewhat ironically, automated theorem-provers were used to produce said proofs since programs could be incredibly large and unmanageable for human practitioners. Program correctness proofs are the primary focus of MacKenzie's *Mechanizing Proof*.

[276]Wos, Winker, "Open Questions Solved with the Assistance of AURA," p. 75.

would, in Wos' design, be tasked with the supply of insights, intuitions, and instincts about the mathematical problems at hand. These would be derived from their experiences with mathematics and from what he took to be the mysterious world of human cognition that would forever elude any attempts at reduction to rule-bound algorithm. The computer would be tasked with the execution of complex and iterative operations that would follow those human intuitions farther, faster, and in different directions than the human user would be able. Built into this vision was a particular vision of proof as constituted by processes of iterative inference punctuated by flashes of intuition. It also entailed a particular division of labor - intuition for the people and inference for the computer. In the remainder of this chapter, I argue that both intuition and inference were *reformalized* in the design and implementation of the AURA program as each was translated into the context of collaborative computing.

Wos accepted a position in Applied Mathematics department at Argonne in 1957 upon leaving the world of academic mathematics. Wos firmly believed (and still believes) that human mathematical practice could not, even in principle, be fully automated. His conviction was based in large part on his personal experiences with mathematical research. His mathematical work seemed to him to proceed by way of seemingly spontaneous moments of insight that were not produced by any process of conscious or rule-abiding reasoning. Wos recounted one story in which the primary realization for his development of so-called "Quad arithmetic" came to him while he was sleeping: "It just came to me. So if you want to give credit, if you want to get pedantic as hell - I've got a big imagination that works when I'm asleep!"[277]

Wos' account of mathematical intuition is reminiscent of Jacques Hadamard's influential essay *The Psychology of Invention in the Mathematical Field:*

[277]Wos interview. Wos also cites anecdotally certain famous historical mathematicians like Henri Poincaré who recounted similar experiences.

> One phenomenon is certain and I can vouch for its absolute certainty: the sudden and immediate appearance of a solution at the very moment of sudden awakening. On being very abruptly awakened by an external noise, a solution long searched for appeared to me at once without the slightest instant of reflection on my part - the fact was remarkable enough to have struck me unforgettably - and in a quite different direction from any of those which I had previously tried to follow.[278]

Hadamard's essay is one of the best known attempts to import insights from psychology, especially about the unconscious, into studies of mathematical discovery. Both Hadamard and Wos called upon their own personal experience as evidence that mathematical problem solving cannot be reduced to intentional and conscious reasoning practices. However, such accounts of personal experience are often influenced by prevalent cultural narratives. Although anecdotal and somewhat folk psychological, these kinds of so-called *Eureka* moments are common tropes in narratives about mathematical discovery. From Archimedes himself, with whom the Eureka moment myth originated to Carl Friedrich Gauss, to William Rowan Hamilton, to Henri Poincaré, and beyond stories abound of sudden insight emerging seemingly from no where when the practitioner was engaged in something other than mathematical thinking.[279]

Though Wos never cites such stories or accounts explicitly, he clearly seems to be

[278] Jacques Haramard, *The Psychology of Invention in the Mathematical Field* (Princeton, NJ: Princeton University Press, 1945). p.8.

[279] This is especially the case in popular histories. The phrase "*Eureka* moment" refers to the Greek Archimedes' reportedly sudden realization of his eponymous principle (also called the first law of hydrostatics). As narrated by Amir Aczel, "*Suddenly it hit him:* the water displaced was equal in volume to that of his body. Famously, he jumped out of the bath and ran naked through the streets of Syracuse shouting, "Eureka, eureka!" (I found it, I found it!)" Aczel, *A Strange Wilderness: The Lives of the Great Mathematicians* (New York, NY: Sterling, 2011): p. 25, my emphasis. Aczel also reports that the famous mathematician Carl Friedrich Gauss penned "EUREKA!" next to a result in a 1796 notebook, Aczel, p. 151. The idea of "eureka" is also often cited in descriptions of William Rowan Hamilton's discovery of quanternions. The podcast *Antimatter* relayed the account as follows: "As regards quaternions, we know exactly when Hamilton had his Eureka moment. According to his own writing, *inspiration struck* on the 16th October in 1843, as he was walking with his wife [...] He was so pleased with the breakthrough that he used his penknife to carve the new equation onto Broom bridge as they past," *Antimatter*, October 16, 2011 (http://coraifeartaigh.wordpress.com/2011/10/16/hamilton-walk-and-maths-week-in-ireland/), my emphasis.

drawing from a body of cultural resources that presents mathematical discovery as consisting in part of such flashes of insight. And Wos brought this perspective to bear on the prospects of automated theorem-proving. Computers would be helpful where rules of deduction and inference and analysis could be made explicit, but they would never be capable of attaining those insights that propel mathematical investigation forward in unpredictable ways. Wos agreed with Newell and Simon that intuition was a central element in mathematical theorem-proving. He disagreed with them and with George Polya whose "heuristics" they borrowed, that those intuitions could be translated into algorithms and transferred to a computing machine.

According to Overbeek, not everyone at Argonne held this position with as much conviction as Wos but most did agree that collaborative software was likely the most promising for approaching open problems in mathematics.[280] More, Wos is a particularly charismatic leader and had a correspondingly central role in shaping the research environment at Argonne until it lost funding in 2006.[281] In line with his view, the base assumption at Argonne was that "Proving theorems in mathematics and in logic is too complex a task for total automation, for it requires insight, deep thought, and much knowledge and experience".[282] So the task of intuiting the way forward in particular proof searches would fall on the human users.

But there was a catch: if human users were going to impart their unautomatable intuitions to the AURA program they needed to put their intuitions into a form that

[280]Overbeek interview, November 2010. They were not the only automated theorem-proving practitioners to hold this position. The development of stand alone theorem-proving programs turned out to be quite difficult, especially for branches of mathematics more complex than the predicate calculus. Increasing effort was directed towards the development of collaborative programs as early at the late 1960s. For example, J.R. Guard, F.C. Aglesby, J.H. Bennett and L.G. Settle, "Semi-Automated Mathematics" in *Automation of Reasoning 2*: 203 - 216; J. Allen, D. Luckham, "An Interactive Theorem-Proving Program" in *Automation of Reasoning 2*: 417 - 434.

[281]The image of Wos as a charismatic and influential director of research at Argonne comes through clearly in the personal accounts of his colleagues in Robert Veroff, ed. *Automated Reasoning and Its Applications: Essays in Honor of Larry Wos* (Cambridge, MA: The MIT Press, 1993).

[282]Wos, Winker. "Open Questions Solved with the Assistance of AURA", p. 74.

the computer could understand and act upon. In order to *implement* a collaborative theorem-proving program, the Argonne team had to fashion an interface for human-computer communication. Intuitions had to be translated into computer *input*. So even though the Argonne team cordoned off intuition as a uniquely human and unautomatable faculty, they still had to rethink intuition with their computer in mind. They had to translate intuition into the technical language of computing. That translation was a *reformalism* - they fashioned intuition into a new form that accommodated the affordances of computing even if it was not produced by them.

The AURA program was constituted by several so-called 'modules' or 'environments', each of which was a programmed subroutine that performed a certain task in the search for a proof.[283] There was one special module whose function was to read in all of the initial input information (called the SYSIN file).[284] This would include a formulation of the problem that the program was going to help solve and the parameters of the mathematical domain from which that problem originated - its axioms and definitions. The SYSIN module was the "first contact" in which a user would present the AURA with a problem. The SYSIN module also afforded users their first opportunity to impart their intuitions about a particular problem to the program.

In particular, users could make use of what was called a "weighting mechanism" - a means for users to translate their ideas about how a certain proof might be found into *quantitative* directions to guide the program's search.[285] Primarily, the computer

[283]Complete specifications are available from Brian Smith, *Reference Manual for the Environmental Theorem Prover: An Incarnation of AURA* (Springfield, VA: National Technical Information Service, U.S. Department of Commerce, 2010 [originally printed by Argonne National Laboratory in March 1988]).

[284]Each input and control specification was coded with the relevant verb first, followed by any extra subparameters the user desired, followed by a semicolon. 'Verb', a term no longer commonly used in computer science, simply denotes what kind of action the computer is to perform on the subparameters that follow. A complete list of the permissible SYSIN verbs is available at Smith, *Reference Manual for the Environmental Theorem Prover*, pp. 14 - 18.

[285]It was designed and implemented specifically by Ross Overbeek as a part of his doctoral dissertation in 1971. Overbeek was already collaborating with Wos and others at Argonne at this time,

would be engaged in the application of rules of inference - deducing consequences from assumptions, axioms, and known theorems using permitted operations of deduction. Weighting was intended to restrict what kinds of deductions and what kinds of consequences the program would prioritize in order to minimize the time spent deducing things that weren't relevant to a particular problem.

With weighting, human collaborators could provide their preliminary sense of what *kind of information* would matter for a given proof. It was a way to communicate to the program that, in deductively exploring a mathematical problem, it should preferentially seek mathematical statements that had particular properties or that it should preferentially employ certain rules of inference. To do this, users would formulate so-called 'weighting templates' that would be included in the SYSIN file. They could indicate, for example, that addition was more important than multiplication, or that the sum of two sums was more important than the sum of two products. Or they could indicate that steps which produced shorter mathematical statements - called clauses - should be prefered to ones.[286] Weighting templates could prioritize the appearance or particular combination of certain terms in a mathematical statement. And so on.

AURA would check whether or not the mathematical statements at play in a given

and he designed the mechanism with their collaborative software development in mind. Overbeek also went on to be one of the central developers of AURA.

[286]"Clauses" were just mathematical statements but with a particular form chosen by the Argonne team for simplicity and standardization. Although AURA could handle problems from many branches of mathematics, the problems would be represented by the predicate calculus. Clause is defined as follows in the manual: "A clause is a logical expression in first-order predicate calculus, extended to include functions, and is the disjunction (primary connective is OR) of primaries known as literals. For our theorem-provers, an example of a clause that is read from an input file is $CL\ q(x,y)\ -p(f(a,b))$ where $q(x,y)$ and $-p(f(a,b))$ are literals, p and q are predicates, $-$ is the negation symbol, f is a function, x and y and implicitly universally quantified variables, and a and b are constants. This clause is thus the logical expression "for all x and y, $q(x,y)$ or *not* $p(f(a,b))$"." (Smith, *Reference Manual for the Environmental Theorem Prover*, p. 3.) Recall from the previous chapter that variables in the predicate calculus can take non-logical assignments, meaning that expressions of this form can be used to represent and explore non-logical branches of mathematics. Taken together, all of the possible deductions for a given problem were called the "clause space." One way to think about Weighting is an attempt to give the computer a "compass" for prioritizing certain paths through the "clause space."

proof search were *instances* of the general form given by a weighting template. For example, $[(x \wedge y) \vee (x \wedge z)]$ is an *instance* of the more general form $A \vee B$. Each weighting template provided AURA with a recursive function for assigning "an integer (weight) to any term (well formed formula) that can occur in a given theorem proving run" depending on whether it contained *instances* of prioritized forms.[287] Those integer values would lead AURA to preferentially pursue those inferences whose ancestral clauses have the highest concentration of relevant information as represented in the templates.

As evidenced by the description of the weighting mechanism in the AURA Reference Manual, these templates and functions were meant to reflect the user's intuitions about a given problem:[288]

> Weighting is the process that assigns measures of complexity to clauses in the clause space. The definition(s) of complexity can be chosen by the user to reflect some *predisposition* that may be, for example, based on an *intuitive notion* of how to direct the proof search. To create new inferences, clauses that seed the next inference... are selected by least weight. Derived inferences that are too complex (too heavy) are rejected.[289]

[287]Veroff, "Appendix A: Term Weighting" in "Canonicalization and Demodulation": 23 - 24, p. 23.

[288]It bears noting that the AURA had a "reference Manual" in the first place. It was circulated internally at Argonne as a technical report, but the distribution list went beyond the automated theorem-proving team. Neither Newell, Shaw and Simon nor Wang drafted reference manuals for the Logic Theory Machine or the Program P. This marks another difference about the AURA program - the hope of the Argonne team was that it would be *used*. Someone other than the development team could, in theory, put the AURA program on their IBM 360 computer and use it in their own problem-solving world. Although AURA's user community remained quite small, later software developed by the Argonne team - especially OTTER and MACE - were circulated quite widely and can still be found in certain variations for online use: http://www.cs.unm.edu/~mccune/otter/. Other large-scale automated tool kits for mathematical problem solving have become nearly omnipresent in technical communities - engineers, computer scientists, physicists, and some mathematicians make regular use of systems like Wolfram's Mathematica, Maple, and Mathlab. These systems, especially the earliest one, called MACSYMA (a system for symbolic mathematical problem solving developed at MIT beginning in the 1960s) will be the subject of some of my future work. I am particularly interested in the role of the User's Manual as a central part of the creation of novel forms of diffuse mathematical communities that were held together not by research questions but by technological infrastructure and shared tools.

[289]Smith, *Reference Manual for the Environmental Theorem Prover*, p. 12. The implementation details for weighting are described on p. 18. My emphasis.

This description of the mechanism breathes "predisposition," "intuitive notions," and the "complexity of clauses in the clause space" in the same breath. Intuition was here translated into an algorithmic, quantitative, and actionable form that could be input to the computer. I contend that these "weighting templates" were quite different from the "flashes of insight" and unautomatable intuitions they were designed to capture. Whatever those insights were and wherever they originated, they ended up being translated into weighting templates, punched onto cards, and were input along with the rest of the AURA SYSIN File.[290]

This was a case of what I call *reformalism*. In fact, it was in considering the transformation of "intuition" in the design and implementation of AURA that I first conceived of the idea that *implementation* was a significant site to explore how computing was made to intervene in knowledge production: in spite of the fact that they wanted to keep intuition out of the computer, reserve it for people, they still had to transform it when time came to make a program that actually ran according to their design and their beliefs. Intuition was here given a particular *form*, namely the "weighting mechanism" and translated into a particular *material*, namely the punched card. This was a new form of intuition, an exercise in fashioning ideas about proof in terms of weighted paths in the set of possible paths to a proof. This was an example of how intuition was understood and enacted differently to make collaboration with computers possible. But this was only one half of the story.

[290] AURA was implemented on an IBM System\360/370 and, for the most part the Argonne team used the punch-card based Assembly Language for those machines to write the program. Assembly language is a low-level (close to the hardware) symbolic representation of instructions and data that users wish to communicate to the computer. See Ned Chapin's 360/370 *Programming in Assembly Language*, second edition (New York: McGraw Hill, 1973 [1968]). The 360 preceded the 370, but with backwards computability so that programs that were run on the former could be run on the latter as well. These models were arguably the most successful of the IBM early commercial mainframe computers. The complete technical specification of the IBM 360 is given in the manual, IBM Corporation, *IBM System\360 Principles of Operation* (1964).

Reformalism II: A Single Step

The weighting mechanism constituted the human contribution to a given human-computer collaborative proof search. Weighting was a way of providing the computer with "shortcuts" through the immense number of inferences permitted in some formal system - a way of avoiding what users thought would be "unprofitable" inferences for a given problem. But there was still inference work to be done and this is where the computer contribution was to be made. The computer would actually go and see what conclusions would eventually emerge from inferring along paths prioritized by weighting templates.

The predicate calculus, which was used to represent problems in the AURA program, came with a set of inference principles for doing that kind of thing.[291] These were laid out in *Principia* and re-presented in a concise and simplified form in Hilbert and Ackermann's canonical *Principles of Mathematical Logic*. They provide "rules for the transformation of logical expressions" and for "deriving the consequences from any premises whatsoever."[292] These rules were thought to capture what Whitehead and Russell were after in *Principia*, namely the "complete enumeration of all the ideas and *steps* in reasoning employed in mathematics."[293] They were rules that described how mathematicians could get from their premises to their conclusions. Sound inference - the logical *step* - the process and the justification for what follows from what, was central to the twentieth-century project of axiomatization and deduction in mathematics.

But the AURA program wasn't going to follow those rules. It's contribution to collaborative theorem-proving was to take different kind of "step" in deducing the

[291]Recall from the discussion in the previous chapter that the variables in predicate calculus can be assigned various values from any branch of mathematics - they can be integers or sets or functions, and so on.
[292]Hilbert, Ackermann, *Principles of Mathematical Logic*, p. 12, p. 101.
[293]Whitehead, Russell, *Principia Mathematica*, p. 3, my emphasis.

consequences of the premises and weighting templates it was given. That step was called the "Resolution Principle" and it was first developed in 1965 by John Alan Robinson. William Miller, director of the Applied Mathematics Division at Argonne, invited Robinson to visit Argonne for six consecutive summers as a visiting researcher. He developed the resolution principle while there, and while in conversation with Larry Wos and other members of the automated theorem-proving group there. But before that, Robinson received his PhD in Philosophy from Princeton University in Philosophy in 1956. Unlike Wang, he did not work explicitly in analytic philosophy, but rather on David Hume's empiricist theory of causation. Like Wang, however, he also developed a dissatisfaction with philosophy that led him into industry where he worked with computers after graduation. In 1960, however, he became a post doctoral fellow at the University of Pittsburgh. While there, he encountered an article documenting an early attempt to use logical principles to prove mathematical theorems.[294] Robinson thought he could do better.

Although Robinson did not focus on logic for his doctoral research, he did have an extensive and also historical training on the subject. Robinson did his undergraduate degree at Cambridge University where he studied Classics and encountered the logic of Aristotle and the Stoics, and also studied more modern logic in the early 1950s with Hilary Putnam who was a member of his dissertation committee.[295] Robinson recognized differences in how logic was done through history, but he proposed that one

[294]The article was martin Martin Davis, Hilary Putnman, "A Computing Procedure for Quantification Theory" in *Journal of the Association for Computing Machinery* Volume 7, No. 3 (1960): 201 - 215.

[295]Biographical information on Robinson is drawn from an autobiographical talk he presented at the 2012 Conference on Automated Deduction in Manchester England; van Emden's oral history with Robinson as published in *Association for Automated Reasoning Newsletter*, No. 89 (October 2010), online edition; and Robinson's autobiographical accounts in "Logic and Logic Programming" in *Communications of the ACM* Vol. 35, No. 3 (March 1992): 41 - 65. Robinson is still alive and his archives remain closed to researchers at this time. I had informal conversations with Robinson at CADE 2012, but for health and travel reasons, Robinson was not able to sit for an oral history with me.

thing was constant: inference steps were designed with human psychology in mind. From Aristotle's syllogism to Hilbert and Ackermann's rules of inference, "steps" in mathematical inference had been designed to accommodate the affordances of human cognition: "Traditionally, a single step in a deduction has been required, for pragmatic and psychological reasons, to be simple enough, broadly speaking, to be apprehended as correct by a human being in a single intellectual act."[296] This was not surprising, Robinson proposed, because deductive proofs were supposed to demonstrate "indubitably" that certain conclusions follow from given premises:

> [E]ach step of a deduction should be indubitable [...] Part of the point, then, of the logical analysis of deductive reasoning has been to reduce complex inferences, which are beyond the capacity of the human mind to grasp as single steps, to chains of simpler inferences, each of which is within the capacity of the human mind to grasp as a single transaction.[297]

The 'basic unit' of inference had historically been the "single intellectual act" of a thinking person. Robinson did not attempt to explain what such an act consisted in or what the intrinsic character and limitations of human cognition were, but he was nonetheless in good intellectual company: modern mathematical logic emerged in the first place as a study of human thinking.[298]

Robinson thought that the advent of digital computing made possible new *kinds* of steps. They were not limited by the same psychological or cognitive characteristics as people - indeed they may not be cognitive or psychological things at all. Robinson wanted to explore how logic could take different forms when it was designed to capitalize on the affordances of computers rather than the limitations of human cognition. He

[296]Robinson, "A Machine-Oriented Logic Based on the Resolution Principle" in *Journal of the Association for Computing Machinery*, Vol. 12, No, 1 (January 1965): 23 - 41, p. 23. He in fact wrote the paper in 1963, but it didn't make it to press until 1965.

[297]Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," p. 23.

[298]Most famous perhaps was George Boole, *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities* (1984).

wanted to devise new processes and new justifications for deducing true consequences from premises:

> When the agent carrying out the application of an inference principle is a modern computing machine, the traditional limitation on the complexity of inference principles is no longer very appropriate. More powerful principles, involving perhaps a much greater amount of combinatorial information-processing for a single application, become a possibility.[299]

Robinson wanted to capitalize on the computer's relative ease with complex operations dealing with large amounts of information in order to make new inference steps possible. These rules would not appear as immediately or obviously justified to even trained human perception. Instead, these rules would be fashioned by human programmers and justified over lengthy demonstrations (indeed, Robinson's introduction of the Resolution Principle occupied twenty-eight pages). But once a principle was shown to be *sound* even if not obviously so to a person, these rules could be executed as a single step by a computer. The computer could infer "further" along a possible deductive path in a single step that a person could in pages of steps. These rules would be difficult for people to understand and impossible for people to use in all the simplest cases. The resolution principle was Robinson's first example of a "computer-oriented" logical principle.

Resolution was a procedure for taking a set of at least two "parent" clauses and combing them in a particular way so as to create new "child" clauses - called the resolvents. The parent clauses had to have a particular feature in order for the resolution principle to be applied: they had to contain what were called "complimentary literals." This feature was defined as follows in the AURA manual: "The condition is that there exist a literal in each clause which, after a consistent replacement of well-formed expressions for the variables in each literal, become identical except for sign which must

[299]Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," p. 24.

be opposite."[300] Two literals are complimentary if, under a certain assignment, one is the negation of the other. Each of the parent clauses must contain such literals. For example, consider the following two clauses: (1) $P(X)$ $Q(X)$, (2) $-P(A)$ $S(X)$. By assigning A to X, the first literal in (1) will be the negation of the first literal in (2).[301] Applying the resolution principle to this pair of clauses would produce a new clause containing all the literals of its parent clauses *except* the complimentary ones. In this example, the two clauses would resolve into (3) $Q(A)$ $S(X)$.

Presented this way, the rule may seem simple enough. However, these literals represent statements in the predicate calculus and these can be arbitrarily long and complex in their own right. More, the resolution principle could be applied to sets of any number of parent clauses, not just two. Still more, applications of the resolution principle checked to see if there were variable assignments that would create the necessary complimentary literals to make them candidates for resolution. The work done by a single application of the resolution principle could require pages of by-hand analysis. The rule in fact was so powerful that "it alone, as a sole inference principle, forms a complete system of first-order logic."[302] That is to say that *every* provable theorem in first-order logic can be proved by the application of the resolution principle alone, where prior formulations involved many rules.

If resolution was so complex that it wasn't easily apprehended "as correct" in a single cognitive act, then how could Robinson claim that it was a sound rule of inference at all? He had to *prove* that it was. In the article where Robinson introduces resolution, he also provides proofs that the resolvents in fact follow from the parent

[300]Smith, *Reference Manual for the Environmental Theorem Prover*, p. 10.

[301]This example is given in Smith, *Reference Manual for the Environmental Theorem Prover*, p. 10.

[302]Robinson, "A Machine-Oriented Logic Based on the Resolution Principle,", p. 24. First-order logic falls in between the propositional calculus and the predicate calculus that we have seen before. It is more complex that propositional calculus because it contains the quantifiers "for all" and "there exists." However, it is simpler than the complete or higher-order predicate calculus because the kinds of values that can be assigned to its variables is restricted.

clauses, that any parent clauses containing complimentary literals can be so-resolved, and so on. Robinson offers traditional, surveyable proofs to justify the resolution principle as a tool for *proving other theorems* by computer. Robinson wrote that "The resolution principle is quite powerful [...] in the psychological sense that it condones single inferences which are often beyond the ability of the human to grasp (other than discursively)."[303] This was an inference principle that people would "grasp" by way of demonstration or, discursive demonstration - by way of proofs, rather than by psychological perception. This "indirect" demonstration is an example of the epistemological significance of automation - proof about inference replaces direct access to inference in understandings and justifications of proof techniques.[304]

In attempting to pass on the underlying intuition of resolution to his students, one computer scientist at the University of Texas at Austin, Gordon S. Novak Jr. created the following example:

> Consider the following axioms:
> 1. All hounds howl at night.
> 2. Anyone who has any cats will not have any mice.
> 3. Light sleepers do not have anything which howls at night.
> 4. John has either a cat or a hound.[305]

Novak demonstrates that one application of the resolution principle produces from these axioms the conclusion "If John is a light sleeper, then John does not have any mice." Resolution works only for "proof by contradiction" in which the conclusion is

[303]Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," p. 24.

[304]Of course, there are those who question the "immediacy" and the "obviousness" of even more basic logical inference. What makes it obvious that the basic rules of inference are sound? What forces someone to adhere to them? The claim that some things just *obviously* follow from others is justified only by that supposed *obviousness*. C.S. Lewis offers a critique of that claim in his delightful piece, "What the Tortoise Said to Achilles" in *Mind* Vol. 4, No 14 (April 1895): 278 - 280. This piece is made famous in Douglas Hofstadter's *Gödel, Escher, Bach: An Eternal Golden Braid* (New York, NY: Basic Books, 1979).

[305]Gordon Novak, "Resolution Example and Exercises," http://www.cs.utexas.edu/users/novak/reso.html.

included as an axiom and it is shown that a contradiction occurs. So in what follows this conclusion will be statement 5., the fifth axiom.

Novak instructs the students that in order to reach this conclusion using the resolution principle, one need only first transform the propositions into a particular *form* - conjunctive normal form - in which only the OR logical operator is used. In that form, the presence of complimentary literals can be seen and the axioms "resolved" in different combinations to achieve eventually the desired conclusion. For this example, the transformation of axioms into well-formed formulas in the predicate calculus and finally to conjunctive normal form can be relatively easily achieved and automated. I will use different notation than Novak for the sake of consistency with notation throughout this dissertation, but the underlying statements are equivalent.

1. $\forall x : HOUND(x) \implies HOWL(x)$
2. $\forall x \forall y : (HAVE(x, y) \wedge (CAT(y)) \implies \neg(\exists z : HAVE(x, z) \wedge MOUSE(z))$
3. $\forall x : LIGHTSLEEPER(x) \implies \neg(\exists y(HAVE(x, y) \wedge HOWL(y)))$
4. $\exists x : HAVE(JOHN, x) \wedge (CAT(x) \vee HOUND(x))$
5. (Conclusion) $LIGHTSLEEPER(JOHN) \implies \neg(\exists z HAVE(JOHN, z) \wedge MOUSE(z))$

These well-formed predicate statements translate into conjunctive normal form as follows:

1. $\neg HOUND(x) \vee HOWL(x)$
2. $\neg HAVE(x, y) \vee \neg CAT(y) \vee \neg HAVE(x, z) \vee (\neg(MOUSE(z))$
3. $\neg LIGHTSLEEPER(x) \vee \neg HAVE(x, y) \vee \neg HOWL(y)$
4. a: $HAVE(JOHN, a)$; b: $CAT(a) \vee HOUND(a)$
5. a: $LIGHTSLEEPER(JOHN)$; b:$HAVE(JOHN, b)$; c: $MOUSE(b)$

The resolution is based on the presence of complimentary literals in parent clauses - that is, two clauses can only be "resolved" if one contains a negation of some literal found in the other. Resolution allows the following new clauses to be created in this way:

- Clauses 1. and 4.b contain the complimentary literal $HOUND$ and $\neg HOUND$ and with resolution they create
  6. $CAT(a) \vee HOWL(a)$ (a new clause without *any* appearance of $HOUND$)
- Clauses 2. and 5.c. contain the complimentary literals $MOUSE$ and $\neg MOUSE$ and can be resolved into
  7. $\neg HAVE(x, y) \vee \neg CAT(y) \vee \neg HAVE(x, b)$
- Clauses 7 and 5.b resolve into
  8. $\neg HAVE(JOHN, y) \vee \neg(CAT(y)$
- Clauses 6. and 8. resolve into
  9. $\neg HAVE(JOHN, a) \vee HOWL(a)$
- Clauses 4.a and 9. resolve into
  10. $HOWL(a)$
- Clauses 3. and 10. resolve into
  11. $\neg LIGHTSLEEPER(x) \vee \neg HAVE(x, a)$
- Clauses 4.a. and 11 resolve into
  12. $\neg LIGHTSLEEPER(JOHN)$
- However we now have a contradiction. 12. amounts to "John is not a light sleeper" but we have from 5.a. that John *is* a light sleeper. So with 5.a. and 12 we have the desired conclusion, demonstrating that the originally assumed conclusion is, in fact, correct.

This example highlights the extent to which resolution is dependent upon *form*. Rather than serving some secondary function, the structure given to logical propositions in this case *makes visible* the element that enables resolution, namely, complimentary literals. Conjunctive normal form was not devised in service of resolution. It was developed earlier in the twentieth century as a means of "simplifying" logic in the

sense that statements in conjuctive normal form have a predictable and repetetive form. This choice of representation enables certain patterns to become visible, like the complimentary literals in logical propositions. However, what is in one sense a "simplification," i.e. a reduction in the number of operators, was, at the same time, a complication. Propositions in conjunctive normal form become as a rule longer and more unwieldily. This is a reason why computers were better suited to exploit the structural features of the notation. When computers apply resolution, they check to see if a given set of propositions can be transformed into clauses in conjunctive normal form containing complimentary literals far far beyond what unaided people can do.

I also think this example highlights the extent to which resolution is not *obvious*. In calling on resolution, I am relying on Robinson's extensive proofs of its correctness as a principle. I know *that* I can take two propositions in the right form and resolve them into a new clause, but I don't know *why* until I survey and study Robinson's proof of its correctness. Of course, mathematicians do this *all the time* - they rely on previous results and incorporate them into their current research. However, they did not do this in *logic*, relative to *inference principles*. Those principles were meant to capture the basic steps of reasoning, our primitive notions of what follows from what which is what was meant to guarantee that the results were correct. Robinson turned to explore different possibilities for logic because he wanted to capitalize on computation not build mathematics up from the bottom according to primitive human cognizing. They could infer more and faster and differently. And more and faster really was different.

Resolution transformed automated theorem-proving: the principle was built in to countless theorem-proving program and many practitioners worked to design variations of the principle for their specific theorem-proving goals. AURA was among the earliest resolution-based programs, and it was the first collaborative theorem-proving program to use the resolution principle. And herein lay what I take to be one final element of

epistemological significance related to AURA: the program would arrive at conclusions or exhibit behavior that the users did not understand. They simply did not know where AURA would end up when "inferring" its way around a problem using resolution. Although the program followed paths given by user weighting templates, it often ended up in places that surprised the users. In using AURA and studying the results, the Argonne team developed a different kind of intuitions and perspectives than the ones they initially set out to preserve. They developed an experimentally and empirically grounded set of intuitions *about resolution* - about how their computer program might behave - rather than merely intuitions about the mathematical problem at hand.

Empirical Intuition: "The Quickest sand Surest Way to Insight"

The Argonne group was committed to an experimental regime. Rather than applying theoretical models of the human mind as Newell and Simon did, or building in a fixed set of logical principles like Wang did, the Argonne group experimented extensively with different test problems to observe how AURA behaved under different conditions. They spent huge amounts of time pouring over AURA's *output* - printed lists of clauses recording what paths AURA took during some proof search. They used these outputs to understand what AURA was doing, but they also used these outputs to try and identify *promising behavior*: they looked for patterns and search paths in the output clauses hoping to isolate what appeared to be fruitful directions of inference for different kinds of problems. They would also identify what they thought were repetitive and uninteresting trajectories in order to retard the engine's pursuit of them.

This empirical and experimental evidence would in turn shape the kind of weighting

templates the users input to the program. This is remarkable because those templates were designed to capture traditional mathematical intuitions about how to proceed in a given proof. Instead, many weighting templates reflected the Argonne team's growing familiarity with the behavior of resolution-based computational inference. These intuitions were grounded in empirical experience with the program. The users' sense of what information to weight emerged from understanding which algorithms cut down clause development in what ways and what patterns were visible in AURA's output on various inputs. Experimentation with and knowledge of the AURA program, rather than the "knowledge and experience" with mathematics provided the content, the structure, and the sources of weighting insights.

For example, one of the earliest strategies developed at Argonne - the Unit Preference Strategy - prioritized shorter clauses over longer ones. Wos presents the strategy in direct response to the nature of clause generation from engines that use the resolution principle. Resolution only permits the inference of new clauses when it is known in advance that the resulting clause was not derivable from either parent clause alone:

> [The Unit Preference Strategy] arises from the fact that the object of the resolution principle is the generation through inferences of two unit clauses which are manifestly contradictory... With this in mind, it seemed worthwhile to orient the program to produce shorter and still shorter clauses in preference to other possible inferences.[306]

After extensive experimentation with resolution-based machines, the Argonne group decided that more productive contradictions were formed between shorter clauses. The human intuition was based on a sense of 'productive contradictions' which was in turn grounded in experimentation and the *a posteriori* analysis of resolution-based behavior.

[306]Wos uses 'contradictory' in the equivalent sense described earlier in which parent clauses for resolution must contain complimentary literals. If one clause contained the L and the other contained not-L (represented with a minus sign as -L), these clauses had the desired character. Wos, "Unit Preference Strategy in Theorem Proving", p. 20.

A second form of intuition could be imparted to AURA in the SYSIN file through the "Set of Support Strategy". This mechanism allowed human users to restrict what sequences of inferences AURA could make.[307] Users could input a list of inferences at least one of which must precede every subsequent inference, as a way of preventing AURA from indiscriminately inferring anything permissible. Through this mechanism, the human user could encode intuitions about what combinations, patterns, or sequences of inferences produced types of clauses that might be relevant to the theorem in question and prevent what was perceived as unproductive swapping of uninteresting equalities. Even more than weighting, such intuitions were tied to extensive experimental knowledge of AURA's runs and which search paths and decision trees resulted from the application of various sequences of inference rules and what kinds of clauses they produced. The weighting strategy and the Set of Support strategy are obviously not the product of traditional mathematical *Eureka* moments. Rather than bringing some human insight into the mathematical problem at hand, users brought insights about *computation* and *AURA's behavior* relative to a mathematical problem.

In some sense, AURA's contributions resemble the preliminary 'scratch work' mathematicians use to approach a new problem. They often try several cases and examples, and search for patterns or useful analogies in order to guide their approach to a proof. By offloading that part of the work to AURA, the resulting human insights were not about the mathematical problem at hand, but about the behavior of a computer program.[308] Here, transformations in mathematical practice were not restricted to the

[307]Smith, *Reference Manual for the Environmental Theorem Prover,* pp. 10 -11.

[308]The *humans+AURA* system qualifies as what Andy Clark and David Chalmers call a "coupled system" of extended cognition: "If, as we confront a task, a part of the world functions as a process which, *were it done in the head*, we would have no hesitation in recognizing as part of the cognitive process, then that part of the world is (so we claim) part of the cognitive process. Cognitive processes ain't all in the head!... In these cases, the human organism is linked with an external entity in a two-way interaction, creating a coupled system that can be seen as a cognitive system in its own right. All components in the system play an active causal role, and they jointly govern behavior in the same sort of way that cognition usually does". Clark, Chalmers. "The Extended Mind" in *The Extended*

process of *implementation*, but continued as users gained experience and new insights from working with the program.

A third form of human-AURA interaction was not part of the SYSIN file. Steve Winker believed that proof-seeking involved more than deductive inferences.[309] Humans, equipped with their unautomatable intuitions, also construct models of problems and search for counter-examples. He therefore devised a way for human users and AURA to collaborate in building simple models of problems to further structure proof searches. It was the addition of this mechanism that lead to AURA's successes in answering open problems from pure mathematics. To understand what is meant by 'model', consider an example from Ternary Boolean Algebra; such algebras are defined by a set of five axioms built with five variables and two functions (one giving the product and the other, the inverse).[310] It was an open problem whether or not any of the axioms were independent - meaning that they could not be derived from any combination of the other four. The fourth and fifth axioms were already known to be *dependent* - meaning they could be derived from the others - but the question remained open for the rest.[311] In order to show the independence of, for example, the first axiom, it was thought sufficient to find values for each variable that satisfied all of the axioms except the first: if the first axiom could be derived from the others,

Mind, ed. Menary. Cambridge: MIT Press, 2010, pp. 27 - 42 at p. 29.

[309]Winker, Steve. "Generation and Verification of Finite Models and Counterexamples Using an Atuomated Theorem Prover Answering Two Open Questions", in Journal of the Association for Computing Machinery, Vol. 29, No 2 (April 1982) pp. 273 - 284.

[310]A list of the axioms is available from Wos, "Solving Open Questions with an Automated Theorem-Proving Program" in Proceedings of the Sixth Conference on Automated Deduction, Vol. 138 (1982) at p. 1.

[311]This result was established by another automated theorem prover designed by J. Allen and D. Luckham. See "An interactive theorem-proving program" in *Machine Intelligence*, Vol. 5 (1970), pp. 321 - 336. Dependence of axioms, it turns out, is quite easy to demonstrate with an automated theorem prover: all you have to do is produce many deductions and check if an axiom is among the conclusions. Independence is much more difficult to establish because we need to show that it is impossible to decude one axiom from the others, which cannot be done by enumerating the infinitely many possible deductions.

whenever they were true, the first must also be true. In this example, Winker wanted human users and AURA to collaboratively develop a model consisting of a revealing domain of values that could be checked for each axiom to see if this was the case.

Winker's model-producing method capitalized on certain features of using AURA that I have identified here: he emphasized that "no properties of the field of mathematics to which the method is applied are used in the method for" producing the model.[312] Humans could have mathematically relevant insights about a problem based on interactions with AURA *without* necessarily having more traditional mathematical knowledge of a problem. The method consisted in three phases, each separated by an exchange of information between the human and the AURA. The fascinating shift in the character of mathematical intuition towards pattern recognition and *a posteriori* structural experimentation is most clear in this design feature.

First, AURA was run on a problem according to the standard protocols to produce many clauses, giving the human user more information to work with. This phase was based on the insight that "mathematical axiom systems are near-minimal... sets that express only the bare necessities to define the mathematical structure under study. Such axiom systems certainly do not include large numbers of extra "interesting" identities".[313] In the first phase AURA provided the user with thousands of printed clauses to peruse in search of ones that appear interesting, useful, or important to the user and again, these value judgements were based on experimental knowledge of AURA's behavior. The human took the insights and equalities gleaned from the first phase and returned to AURA a partial model for testing. AURA then performed a second run, this time incorporating the the incomplete model and output all those equalities

[312]Winker, "Generation and Verification of Finite Models and Counterexamples Using an Automated Theorem Prover Answering Two Open Questions", p. 274.

[313]Winker, "Generation and Verification of Finite Models and Counterexamples Using an Automated Theorem Prover", p. 278.

or formulas that pertain to the missing parts. Again, the user studied thousands of printed clauses looking for patterns and potentially useful identities in order to complete the model. AURA then, without further human input, checked every instance of the model. For the case of Ternary Boolean Algebras, AURA checked that every instance of the model satisfied all axioms except the first. If the verification run was successful, and in this case it was, the desired theorem would be proved. Therefore, in 1978 *humans+AURA* proved that, indeed, the first three axioms of Ternary Boolean Algebras are independent, thus making their first novel contribution to mathematical knowledge.

Each instance of human input based on the Argonne team's idea of what was interesting, what was potentially and important, and what might work - weighting, Set of Support, and model construction - was based on extensive experimentation and intimate knowledge of the lists of clauses AURA output on various previous runs. Ross Overbeek's made the following revealing statement about the Argonne group's methodology:

> I began to realize that Wos took experimentation very seriously... Most of our work together focused on laying out a set of problems that could not be solved, examining thousands of output clauses to determine what was going wrong, designing algorithms to correct the situation, and then observing the result. This is the first and most central aspect of Argonne culture: ... Rule 1. Run experiments and observe what is going wrong. *It's the quickest and surest way to insight.*[314]

Furthermore, Overbeek acknowledged that the kind of intuition needed to successfully collaborate with AURA was not simply or obviously logical or mathematical intuition. In fact, he proposed that "the trained logicians, with the exception of George Robinson, really had little idea of how the proposed algorithms actually worked (or why they did

[314]Overbeek, "Wos and Automated Deduction at ANL: The Ethos" in *Automated Reasoning and Its Applications; Essays in Honor of Larry Wos*, ed. Robert Veroff, Gail Pieper. Cambridge: MIT Press, 1997, pp. 1 - 12 at pp. 2 - 3. My emphasis.

not work)".[315] This was not traditional mathematical intuition; it was technological and computational intuition, grounded in experimental experience and *a posteriori* revelation. And yet, this new collaborative and experimental method led to solutions of open mathematical problems.

Conclusions: Open Problems

Throughout the dissertation, I have argued that implementation has epistemological significance. In designing computer programs for proving theorems and actually making them run, new mathematical knowledge known in new ways emerge. These are associated with the practices and perspectives that came out of reconceptualizing elements of mathematics through the lens of computing. The theorem-proving programs I discussed in previous chapters, however, only proved theorems that were already known to be theorems. The AURA, however, worked with human users to provide proofs for previously open problems. And unlike, for example, the computer-assisted proof of the Four Color Theorem, it was not known in advance what contribution the computer would ultimately make to these new solutions.

Upon receiving the "Award for Current Research in Automated Theorem-Proving," Wos and Winker recounted their experience working with AURA to solve two open problems. The anecdotes, taken from Finite Semigroups and Equivalential Calculus, were designed to demonstrate that AURA could enable and surprise its users beyond what traditional mathematicians might expect. In the case of Finite Semigroups, the Argonne team knew so little that they misunderstood an open problem completely and in the first instance ended up proving the wrong thing all together.[316] After realizing

[315] Personal email from Overbeek, November 2010.
[316] Wos, Winker; "Open Questions Solved with the Assistance of AURA", pp. 75 - 77.

this, they went on to work with AURA to solve the real problem in 1981.[317] They used this case to emphasize that AURA had enabled them to work in fields beyond their knowledge. In the case of Equivalential Calculus, after many unsuccessful runs, Brian Smith attempted to *force* AURA to recreate a *known* proof of some theorem using weighting. Instead, AURA went ahead and "found a proof half as long".[318] Even the heavy-handed imposition of the user's knowledge could lead to surprising results.[319] The Argonne team, in spite of always emphasizing the necessity of human insight and experience in the work of mathematics, proposed that AURA's assistance made possible new and surprising possibilities for the work of proof. Not just implementation but also *use* had transformative epistemological significance. Wos has even indicated that, if it weren't for the fact that he may not have been taken seriously by journal editors, he would have liked to include the name of his Automated Reasoning engines as co-authors of produced proofs.[320]

However, the Argonne group did not justify AURA's value merely in terms of the theorems they proved with it. Similarly, AURA's relevance to the history of mathematics is not in its contributing grand theorems to the corpus of mathematical truths. Instead, the Argonne team proposed that a new way of contributing to mathematical knowledge was made possible through collaboration with their unconventional assistant. AURA's significance to the history of mathematics is similarly grounded in its revealing how the image of mathematics, the practice of mathematics and the way in which mathematical problems are understood and solved are constantly negotiated within the material culture and practices of mathematical communities. I have also

[317]Winker, Wos; "Semigroups, antiautomorphisms, and involutions: a computer solution to an open problem, I," in *Mathematics of Computation*, Vol. 37 (1981), pp. 533 - 545.

[318]Wos, Winker; "Open Questions Solved with the Assistance of AURA", p. 78.

[319]Hans Jörg Rheinberger's concept of an "experimental system" is meant to capture precisely how very disciplined scientific instruments can still produce surprises in the making of scientific knowledge. See *Toward a History of Experimental Things* (Stanford University Press, 1997).

[320]Wos interview.

argued that mathematical intuition took on new form and new content - intuitions trained by working with software and about that software's behavior - were introduced as possibilities for theorem-proving.

Conclusion: Reflections on the Aftermath

Intuition and Inference: (Re)configuring Minds and Computing

To compare people to computers, even to say that they are different, transforms the terms within which both are understood. Lucy Suchman made this point in *Human-Machine Reconfigurations*, relative to the development of emotive robotics. Following the work of Monica Casper and Sara Ahmed, she would challenge any account that seeks to distinguish "human" from "nonhuman" *in advance* of historical or sociological analysis:

> [E]mpirical investigations of the concrete practices through which categories of human and nonhuman are mobilized become salient within particular fields of action. And in thinking through relations of sameness and difference more broadly, Ahmed (1998) proposes a shift from a concern with these questions as something to be settled once and for all to the occasioned inquiry of "which differences matter, here."[321]

[321]Suchman, *Human-Machine Reconfigurations*, pp. 1 - 2. Casper, Monica. "Reframing and grounding nonhuman agency: What makes a fetus an agent" in *American Behavioral Scientist,* Vol. 37 (1994): 839 - 856. Sara Ahmed, *Differences that Matter: Feminist Theory and Postmodernism* (Cambridge UK: Cambridge University Press, 1998).

Following Suchman who inspired the subtitle of this dissertation, I ask - which differences mattered *here*, in this particular field of action, in automated theorem-proving? How were humans and computers understood relative to one another in the hands of those who sought to automate mathematical proof? What categories of human and nonhuman were mobilized and salient within attempts to automate the work of proof? My actors disagreed, as we have seen, about how and how much computers resembled their human counterparts in pursuing the work of proof. But their disagreement went deeper than that - in establishing "sameness" or "difference" between people and computers, they fashioned each differently, according to particular, carefully chosen theories of human faculties and machine possibilities.

For Newell and Simon, if computers were going to be admitted as agents of proof, they would have to exhibit the same faculties that supported human mathematical agency. The lens of systems analysis, a dominant research paradigm at RAND, enabled them to perceive a fundamental sameness between human minds and digital computers. For them, both were "species of the genus information processor" - systematic, algorithmic manipulators of symbolic information.[322] Human mathematical agency was constituted by faculties of deductive reasoning and heuristic search, each construed as a formalizable rule-bound process. If they could identify those rules of reason and those heuristics, they could program a computer to perform the very same.

Newell and Simon were not concerned that this problem-solving behavior would be made manifest in quite different material substrata - brains and computers, that is. They focused on higher-level *formal* behavior as the fundamental seat of reasoning and intuition and they believed that this formal faculty could be shared between human and machine. In fashioning human-computer similarity, Newell and Simon de-emphasized

[322]Newell, Simon, *Human Problem Solving.* Vol. 104, No. 8. Englewood Cliffs, NJ: Prentice-Hall, 1972 as cited in Heyck, "Defining the Computer, Part I," p. 43.

human bodies and brains and lived experiences. The faculties of mathematical intuition and imagination, associated by others like Wos with embodied experience, irrationality, or the unconscious, were for Newell and Simon exercises in rule-bound information processing. Although this vision of mathematical reasoning as a disembodied, rule-bound, symbolic enterprise has a history dating back to the nineteenth century at least, it was given new formulations in the context of computation. Brains became the "hardware" in which the mind - associated with "programs" or "software" - was realized. The overwhelming material differences between human brains and bodies and the mammoth metal machines that constituted 1950s computing were quieted through the characterization of reasoning as an abstract formal activity.

Hao Wang did not see a fundamental sameness between human reason and computation. He didn't just disagree with the comparison, he disagreed with the formulation of human faculties that made it possible. He didn't recognize human intuition and human practice in Newell and Simon's automatable heuristics. Wang believed that human mathematical faculties were only partially, rather than entirely, rule bound. Some of what we do could be reduced to algorithmic processes and subsequently automated, but other parts of what we do could not.[323] For Wang, human thinking was a contingent and complicated faculty, shaped by the particular experiences and circumstances of the thinker. In 1983, upon receipt of the AMS Milestone award for automated theorem-proving, he remarked that "[a] particular act of recognition or reasoning depends on one's genes, the history of one's mind and body, some essential relation to desire, even race, class, sex and many other factors. Of most of these factors we have at best an

[323]It is perhaps worth noting that this position resembles, and precedes by a few decades, sociologist of science Harry Collins' criticism of Artificial Intelligence - that computers will only be capable of intelligence in those domains where human activity has already been mechanized, like calculation. Other domains where tacit, embodied, and deeply social dimensions constitute human knowledge, will prohibit automation. Collins, *Artificial Experts: Social Knowledge and Intelligent Machines.*

imperfect knowledge and understanding."[324] Human reasoning could not be reduced to a universal formalism because each person's reasoning faculty develops uniquely according to numerous, only partially understood factors. What Wang recognized in computation was instead an *ideal* human reasoning agent, one historically imagined by logicians, philosophers, and mathematicians to proceed according to formality alone. He faulted Newell and Simon for accepting as truth the myth that human mathematical reasoning was rule-bound, algorithmic, disembodied. What they automated was an abstract model of the mathematician that resembled actual mathematicians just as much as computers resembled brains.

Wang fashioned *difference* between people and computers on quantitative and material grounds. Even where human practices and faculties *did* operate according to some automatable rule, computers would be able to follow more of those rules faster and further and more efficiently than any person could. And for Wang, more was different - this speed and efficiency opened up new possibilities for theorem-proving agency. In his fashioning, the materiality of people mattered - it shaped their reasoning faculties. And the materiality of computers mattered too - the technological specificity of modern computers afforded them speed and efficiency that would be impossible for an unaided person.

From where Wang stood, there was a deep difference between humans and machines tied to the different possibilities that different forms of embodiment could afford. Although they would never reason like people or prove theorems like people, computers could approximate an ideal of the mathematician, leaving the real ones behind, and in so doing become powerful but quite different agents of proof.

[324]Wang, "Computer Theorem Proving and Artificial Intelligence" in *Automated Theorem Proving: After 25 Years*, p. 55.

The approach of the Argonne group represented a kind of synthesis to the "thesis" and "antithesis" presented by Newell-Simon and Wang. They agreed with the former that intuition was a fundamental element of mathematical research, but they disagreed that it could be automated. They agreed with Wang that computers, by following rules faster and further than people, opened up new and powerful possibilities for deduction, but they disagreed that such methods could constitute *proof* on their own, in the absence of human intuition and understanding.

Larry Wos, accordingly directed early automated theorem-proving research at Argonne toward the development of collaborative software that would incorporate both human intuitions and computational power. Like Wang, they primarily emphasized *difference* between human and machine - associating unautomatable intuitions with the former and fast, efficient deduction with the latter. But in fashioning these faculties so as to emphasize difference, they posited a *third agent* - the hybrid collaborative human-computer agent that would be capable of proving theorems in ways that would be impossible for each on their own.[325] They parsed the work of proof into two components - intuition, reserved for human users, and inference, assigned to computers, which together would constitute a superior and quite new form of collaborative and hybrid agency of proof.

In each case, the prospect of automation prompted practitioners to reflect on the character of human faculties through the lens of computation. Practitioners identified faculties like mathematical intuition and deductive reasoning as either rule-bound or not, embodied or not, unique or not. Their questions were as new as computers. That is to say they were at once very new and very old. Recent histories of computing walk a precarious line, emphasizing on the one hand, that computers did not represent a decisive break from the machines and practices that preceded them, while on the

[325]I explore this hybrid agency in Dick, "AfterMath."

other hand, admitting significant novelty. Similarly, questions about the sameness and difference that obtains between people and computers took up centuries-old threads dating back to debates about mechanism, determinism, and mind-body dualism.

In another sense, these questions were very new. This was in part given the novel metaphorical, practical, and technical resources - like programming languages, hardware and software, studies of algorithm and complexity - that were mobilized in service of these debates where computers were involved. Where human faculties were seen as of-a-kind with computation, they had to be understood in these terms and translated into these tools. And where human faculties were being opposed to computation, they had to be defined with the scepter of this quite particular "other."

The practitioners I studied in this dissertation didn't just disagree about how much computers were like people. While fashioning sameness and difference between them, these practitioners emphasized different characteristics - materialities and formalisms, speeds and temporalities, rules and indeterminacies. Not only did I want to understand the differences and samenesses that my actors carved out between people and machines and theorem-provers, I wanted to know how these were in turn translated into their actual software development. I wanted to know what they *did* with these configurations, what new practices and tools they introduced to proof in tandem.

Practitioners of automated theorem-proving reflected upon the character of human mathematical faculties through the lens of computation, in order to fashion sameness and/or difference between them. In this way those faculties were (re)configured - parsed in new ways, understood through new metaphors, held together or apart from elements of the new technology. But that technology was (re)configured in tandem, as practitioners developed quite different sets of tools and techniques to embody their views. I have argued that you could not translate views about the relationship of people and machines into running programs without transforming them yet again.

Consider for example, the very different ideas of human intuition that were at work in the Logic Theory Machine and the AURA. Both development teams wanted to *preserve* human intuition, unchanged. In the case of the former, Newell and Simon sought to identify the heuristic rules that lay at the heart of human intuition and automate them. The Argonne group who developed AURA did not believe that intuition could be so automated, and instead wanted to provide human intuitions to computers from the outside to guide their deductive search. A particular vision of human intuition informed each project, and both wanted to keep that vision in tact in the context of computer proof. But in each case, human intuition in fact was given a quite different form, visible only when one attends to often neglected process of *implementation.*

In the case of the Logic Theory Machine, Newell, Simon, and Shaw developed a whole new programming language thought to capture the kind of "symbolic information processing" at work in human minds. In that language, as I discussed in Chapter One, objects were represented in a particular way - namely as "linked list information structures" - and manipulated according to particular processes - namely, list processing operations. This formal and material system, I argued, represented a very decisive break from the paper-based representations and operations with which human logicians worked on logical proof. The system introduced quite foreign processes and structures to the logic of Russell and Whitehead that Newell and Simon sought to preserve in automation.

Wos and his team also sought to preserve the character of human intuition in developing the AURA. Unlike Newell and Simon, however, they did not believe that intuition could be reduced to a set of rules and automated. Instead, they cordoned off intuition as a uniquely human faculty. They wanted human users to provide their unautomatable intuitions to AURA to guide the latter's deductive search, prioritizing certain inferential paths over others. But even here, where they did not seek to au-

tomate intuition at all, that faculty was still given a very new formulation. If human users were going to collaborate with the AURA, if they were going to provide it with their intuitions, these latter had to be transformed into *input*. Intuitions had to be translated into a form that the computer could understand. As discussed in Chapter Three, Ross Overbeek developed what was called a "weighting mechanism" by which users could input templates for patterns of inference and forms of logical proposition that were to be prioritized or avoided. These weighting templates were a far cry from the uniquely human, unautomtable "Eureka" moments that Wos sought to preserve. They represented a translation of that view into an actionable computational tool, part of an interface that enabled a particular form of human-computer collaboration in the work of proof.

In a sense, Newell-Simon and the Argonne team operated with opposite views of intuition. For the former, it was rule-bound, automatable, heuristic search. For the latter, it was unautomatable, embodied, and uniquely human. Each devised a set of computational tools to put those visions of intuition to work in the context of automated theorem-proving. In both cases, the computational tools developed diverged in quite significant ways from the vision of human intuition that preceded it. In both cases, that vision of human intuition had to be made to accommodate the limitations and constraints of computation. That translation was always transformative, even in cases like this where practitioners sought to preserve something human.

Interesting transformations also took place where practitioners sought instead to harness the power, speed, and efficacy of computation for something decidedly *not* human. Both Hao Wang and John Alan Robinson were interested in what computers made possible for proof that would be impossible for their human counterparts. In developing the Program P, Wang worked with an existing theorem from mathematical logic, Herbrand's Theorem. That theorem, as seen in Chapter Two, established a

relationship between propositional logic and the more complex predicate logic, giving an algorithm for proving theorems from the latter using only tools from the former. However, it was *impossible* to actually use this algorithm to prove theorems because it would require the execution of an infinite number of steps. Only some "ideal" mathematician that could go one applying the theorem's dictates forever would be able to actually prove theorems this way.

The theorem instead offered a particular insight - a way of understanding two branches of logic in terms of one another. Wang believed that computers better approximated that "ideal" mathematician better than any person. He took Herbrand's theorem - an "in principle" algorithm - and he transformed it into a set of computational tools. The Program P deployed methods of so-called pattern recognition by sequential tables that produced actual proofs of predicate theorems in terms of the infrastructure of propositional calculus, methods that were beyond the reach of unaided human mathematicians who could not execute enough steps to realize them. Here, the epistemological status of Herbrand's theorem was transformed. The theorem went from being a kind of "meta" insight about the relationship between two branches of logic to an actionable tool kit for producing proofs. It went from being an imaginary algorithm to an actual set of computational operations. With it, the ideal mathematician who could go on forever, was partially actualized by a machine that could go on much further and faster than any person could.

John Alan Robinson also sought to exceed perceived human limitations with computers, and precipitated an epistemological transformation as a result. He proposed that, through all of history, the rules of logical inference, the deductive scaffolding that held logical formalism together, were decisively human-oriented. Those rules were meant to capture steps for moving from something true to something else true in a manner that could be perceived as correct by human minds, with all of their particular

psychological and combinatorial limitations.

Computers did not share these limitations and neither, Robinson believed, should their logics. He set to work devising computer-oriented logics with new inference rules that may not be perceptible to people, but that computers could easily execute. The Resolution Principle was just such a rule. As discussed in Chapter Three, it was powerful enough to replace all of the inference rules in *Principia Mathematica*, sufficing alone as the mechanism for establishing the theorems in those volumes. Resolution enabled computers to find more efficient paths through the sets of permissible inferences, finding shorter routes to desired theorems. But resolution was hard for humans to follow. It was too complex to be perceived as intuitively or obviously correct by human logicians. And the proofs it produced quickly became intractable as well.

In developing the Resolution Principle, Robinson precipitated a transformation of *logic* itself. Logic was once the project of identifying and capturing the basic units of *human thought*. George Boole, a founding father of the discipline, in fact took these to be one and the same, as evidenced by the title of his most foundational work - *An Investigation of the Laws of Thought, On Which are Founded the Mathematical Theories of Logic and Probabilities*.

When Russell and Whitehead elected logic as the foundation for all of mathematics, within which its truths could be established, they too wanted to make explicit the "*primitive* ideas" and the "simplest and most convenient notation" in order, ultimately, to enable the "complete enumeration of all the ideas and steps in reasoning employed in mathematics."[326] They didn't want the most economical or powerful steps for reasoning but the ones that captured and made clear what right human logical reasoning looks like and how that accounts for the truths of mathematics. Resolution, and other computer-oriented logics like it, were not primitive, they weren't simple, and

[326]Whitehead, Russell, *Principia Mathematica*, Volume One, pp. 1 - 3.

they weren't convenient for human perception. Where they were developed and set in motion, logic was disassociated from human thinking.

This disassociation had an institutional dimension as well. Resolution-based proofs were not recognized, for obvious reasons, as part of Artificial Intelligence research. Automated theorem-proving based on formal logics that did not necessarily accommodate human thinking was "found lacking" by the Artificial Intelligence Community.[327] As such, different designators were chosen - especially "Automated Deduction" and "Automated Reasoning" - under which resolution and other computer-oriented studies of logic would be united. Journals and conferences were created for the collection and distribution of work in these fields. Today, most automated theorem-proving work is presented at the Conference on Automated Deduction (CADE) and the International Joint Conference on Automated Reasoning (IJCAR).

Resolution: (Re)configuring Proof

In 2012, I attended the annual CADE in Manchester, England. Each year, in conjunction with that conference a tournament for automated theorem-proving programs is held: The CADE ATP System Competition, World Championship for Automated Theorem Proving (CASC).[328] CASC was created in 1997 by Geoff Sutcliffe and Christian Suttner who remain its organizers today. The competition is housed each year somewhere at the CADE conference venue and runs alongside presentations and panels until an award ceremony concludes the conference. The competition has a playful feel, with the development teams of different systems asking each other questions about design choice as they watch the results come in. As shown in Figure 4.1, the results come in the guise of horse race iconography displayed on a TV monitor.

[327] Overbeek, Lusk, "Wos and Automated Deduction at ANL," p. 6.
[328] The tournament website: http://www.cs.miami.edu/~tptp/CASC/.

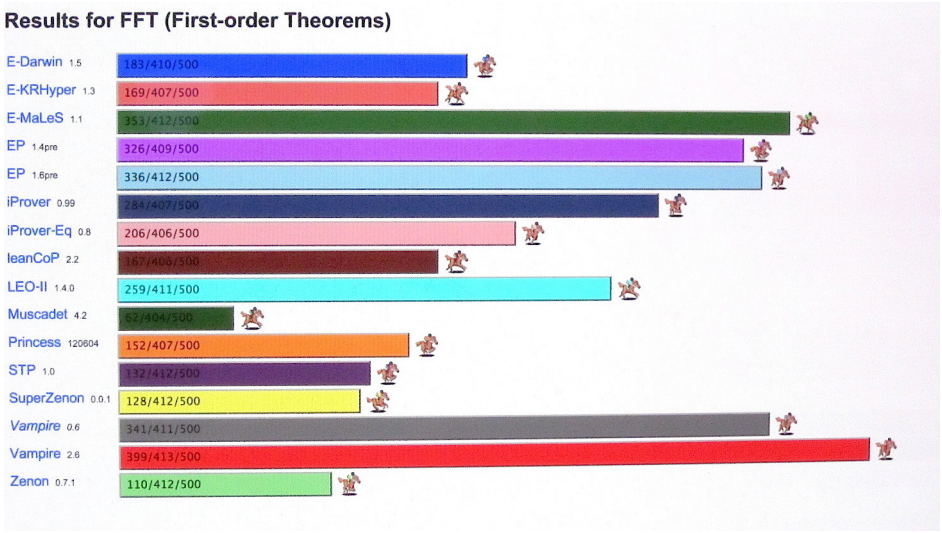


Figure 4.1: CASC 2012 Results Display. My photograph. Manchester, UK, 2012.

Depicted in Figure 4.1 are the results of the "first-order logic" (or predicate calculus) theorem proving category at the CADE 2012 CASC tournament. Each program is represented by a racing horse figure and its progress is measured by a horizontal colored bar.

Progress in this tournament is measured relative to a standardized body of benchmark problems called the TPTP Problem Library, for "Thousands of Problems for Theorem Provers."[329] The library was created in 1994, in an attempt to "move the testing and evaluation of ATP systems from the present ad hoc situation onto a firm footing."[330] As we have seen in previous chapters, some problems are harder than others, and some have more revealing structural properties as well.

The TPTP Library aimed to provide a shared set of benchmarks that would actually reveal "the intrinsic power" of a given theorem-proving program rather than the idiosyncratic results that make their way into published presentations of theorem

[329]See Geoff Sutcliffe, Christian Suttner, Theodor Yemenis, "The TPTP Problem Library" in *Proceedings of the 12th Conference in Automated Deduction* (1994): 252 - 266.
[330]Sutcliffe, Suttner, Yemenis, "The TPTP Problem Library," p. 252.

proving programs.[331] TPTP continues to grow and change over time, preventing a stagnation of theorem-proving system development and also now accepts problems from outside parties as well. For example, some problems come from industry - as long as a problem is voided of its content and presented in the "unambiguous format" of problems in the library, it can be included and subjected to the efforts of participating theorem-proving programs. The horse race bars indicate how many of the TPTP library a the associated program has solved (if it was able to solve them) and in how much time.

The tournament has traces of Argonne everywhere. The benchmark "programs to beat" at CASC well into the twenty-first century were Argonne theorem-proving programs OTTER and the EQP (for Equational Prover) which both borrowed insights and infrastructure from AURA.[332] More, the TPTP Problem Library was inspired by earlier efforts of the Argonne team, especially Wos, to identify and standardize problems for comparing the merits of theorem-proving systems.[333] But most especially, the majority of the programs represented here, and indeed many theorem-proving programs being developed today, incorporate variations of Robinson's Resolution Principle.

Resolution turned out to be an incredibly powerful inference rule and incredibly effective for use in computer deduction systems. The pages of the *Journal of Automated Reasoning* filled up with variations, improvements, and expansions of resolution, and power resolution-based theorem provers emerged around the U.S. and Europe.[334]

[331]Sutcliffe, Suttner, Yemenis, "The TPTP Problem Library," p. 252.

[332]Both were developed under the direction of William McCune, who was just starting out at Argonne when AURA was being developed in the late 1970s. The EQP is famous for having produced a solution to a previously open problem concerning the axioms of Boolean Algebra - namely Robbins' Conjecture. See McCune, "Solution of the Robbins Problem" in *Journal of Automated Reasoning* Vol. 19, No. 3 (1997): 263 - 276.

[333]J. McCharen, A. Overbeek, L. Wos, "Problems and Experiments for and with Automated Theorem-Proving Programs" in *IEEE Transactions on Computers*, C-25, No. 8 *August 1976): 773 - 782.

[334]There have been on the order of 500 articles on resolution published in the *Journal of Automated Reasoning* since its founding in 1983.

Recall from Chapter Three that resolution was designed to capitalize on the affordances of computers, not to accommodate the restrictions of human psychology. In the AURA program, however, the power of resolution remained tethered by the real-time interventions of a human user. Increasingly, resolution provers worked with resolution and its variations alone. Or, as was explained to me Lawrence Paulson, many contemporary programs today collaborate *with another* program that specialized in a complimentary kind of problem-solving rather than a human user, aggregating their complexity and their power increasingly beyond human tractability.[335]

Resolution took theorem-proving further and further from human view. Resolution provers were using powerful tools of deduction and reasoning to process enormous data bases, solve complex industry problems, prove the correctness or disfunction of other programs, and to prove theorems in mathematics. As these programs became more and more powerful, the human ability to recover and reconstruct their behavior and to understand their results waned. Human intuition and direct understanding became increasingly removed from the work of automated theorem-proving, which became increasingly black-boxed.

The 2012 Conference on Automated Deduction was a particularly well-attended affair because it was held in conjunction with the headlining celebration of the centenary of Alan Turing's birth. Alan Turing who, in 1948, imagined the possibility of disembodied brains engaging in mathematical theorem-proving. Part of the draw of the conference was that a number of significant practitioners were invited to share their reflections on the history and state of the field. Among them was John Alan

[335]Paulson is the original developer of a theorem prover called Isabelle. It is an interactive prover, though primarily it collaborates with other programs designed to maximize performance in complimentary inference practices. Isabelle's main infrastructure is characterized by a version of the Resolution Principle. See Paulson, "The Foundation of a Generic Theorem Prover" in *Journal of Automated Reasoning*, Vol. 5, No. 3 (September 1989): 363 - 397. The most recent variation of Isabelle is also described in Paulson, *The Isabelle Reference Manual*, July 2008: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.9052&rep=rep1&type=pdf.

Robinson. He stood before a room full of automated theorem-proving researchers and, to my astonishment, said that in fact "formalization is not the thing we do when we are interested in the *truth*."[336]

In the 1960s, Robinson sought to surpass human mathematical agency with machine-oriented logic, justified by the infrastructure of formal logic alone, not by human perceptions, intuitions, or cognitions. In 2012 Robinson was questioning what Resolution-style formalization really offered. In his talk, he returned often to the question, "what are proofs really *for*?" Instead of citing other formal logicians in this talk Robinson shared what he thought were the lessons of the French conglomerate Bourbaki, Luitzen Egbertus Jan Brouwer, Ludwig Wittgenstein, and Imre Lakatos who stood in opposition to Hilbert's program of complete axiomatization and formalization in mathematics.

In different ways, each of those mathematicians differentiated between a demonstration *that* something is true and *why* something is true. Reflecting on their positions Robinson said "It's no good just following step by step a proof and saying ok, ok, ok, if you don't really have understanding, if you don't really know what's going on. [...] You haven't really understood a proof if all you did is verify it. [...] It's only if you look into the *ideas* under that sequence of inference [that you] have you understood the proof."[337] Robinson proposed to a room full of resolution researchers that it was to those ideas and insights and to human understanding that they should turn their automated theorem-proving efforts - that is "where we should go next."

This was a quite different perspective compared with his earlier program of advocating for a new logic, new principles and practices of inference that capitalized on the

[336]John Alan Robinson, "Davis, Putnam, UNIVAC, FORTRAN, and other miracles: How I got involved with Computational Logic and What Happened Next," delivered at the International Joint Conference on Automated Reasoning, CADE invited Speaker, Friday June 29, 2012.

[337]Robinson, "Davis, Putnam, UNIVAC, FORTRAN, and other miracles." He made these comments beginning in minute 74 of the talk.

power of computer, *eschewing* the limits of human understanding, human intuition, and human psychology. In 2012, he capitulated that some of the critics of Resolution did in fact "have something" when they warned that something important was lost in the offloading of theorem-proving to processes of inference that were bereft of human insight.

One formulation of this criticism was proposed by mathematician Michael Harris in "Do Androids Prove Theorems in Their Sleep?"[338] Harris writes that to computers, every "step" is the same. Computers cannot, unless directed by a person, identify those "steps" in a proof that contain what he calls the *key* - the particular insights that capture the *why* rather than the *that* of a theorem's truth. He studies how mathematicians identify in their demonstrative expositions some "key" or "main" or "crucial" or "fundamental" or "essential" insight that "hints that mathematical arguments admit not only the linear reading that conforms to logical deduction but also a topographical reading that more closely imitates the process of conception."[339] He cites David Byrne's dictum that "heaven is a place where nothing ever happens" to introduce the world of the computer were all steps are created equal, all inferences are merely steps, and computers cannot show people what "key" insight grounds the *truth* or reveals the *why* of a mathematical theorem.[340]

The very possibility of automated theorem-proving, according to Harris, is predicated on the belief that "nothing really happens when a theorem is proved."[341] And this is a common criticism among mathematicians and philosophers - computers simply can't differentiate between the steps that "matter" and the steps that "don't matter."

[338]Harris, "Do Androids Prove Theorems in Their Sleep," in *Circles Disturbed*, eds. A. Doxiadis, B. Mazur, (Princeton, NJ: Princeton University Press, 2012): 130- 182.
[339]Harris, "Do Androids Prove Theorems in their Sleep?" p. 133.
[340]Harris, "Do Androids Prove Theorems," p. 135.
[341]Harris, "Do Androids Prove Theorems," p. 136.

They can therefore only show *that* something is true but not *why* it is.[342]

Robinson invoked a debate from earlier in the twentieth century - between Hilbert who he called "Mr. Formalization" and Brouwer, who opposed the project of formal axiomatic proof to articulate his change of heart. For Brouwer, "mathematics is a creation of the mind" and the "truth of a mathematical statement can only be conceived via a mental construction that proves it to be true" and as such, mere deduction and thoughtless automated deduction at that, could never in fact establish the truth of a mathematical statement.

I see traces of something even older at work in the association of mathematical truth with the work of human cognition. A related position can be identified even in Aristotle who proposes that *"thought* is an *energeia* - namely a process that *actualizes* what exists only in potentiality in the premises of a mathematical system.[343] That is to say - it is the *thinking* that introduces mathematical truths to the world even though they are contained in potentiality within the axioms of some formal system.

Debates abut the role of human intuition and of formalization in proof is old.

[342]The belief that mathematics "reduces to an immense tautology", that its truths are always already there, present in the axioms, needing only to be mechanically teased out, has been criticized from a number of other angles in philosophy of mathematics (I borrow the phrase "reduces to an immense tautology" from Stuart Shanker, in his analysis of Wittgenstein's philosophy of proof. Shanker, *Wittgenstein and the Turning Point in the Philosophy of Mathematics*, p. 82). Wittgenstein, for example, emphasizes the *normative* role of proofs in establishing new "grammatical" rules to follow in certain domains of mathematics. See Wittgenstein, *Remarks of the Foundations of Mathematics Remarks on the Foundation of Mathematics*, ed. G. H. Wright, R. Rhees, and G. E. M. Anscombe; trans. G. E. M. Anscombe (Oxford, UK: Blackwell Publishing, 2001 [1956]), III). Or, Imre Lakatos has emphasized the constant reconstruction of boundaries around the class of objects or problem about which something is being proved: whenever possible, mathematicians will define possible counter-examples away rather than adopt a new approach to the proof all together (*Proofs and Refutations*, p. 10). The actual construction of proofs by people in history serves at once to adjudicate what will count as truth, and what objects, problems, and properties are included and excluded from that designation. These are just two accounts of the *work* done by proofs and by theorem-provers that raise questions for automation.

[343]My understanding of Aristotle's position in this regard is due in large part to G.E.R. Lloyd, "Mathematics and Narrative: An Aristotelian Perspective" in *Circles Disturbed*: 389 - 406. I am also indebted to discussions with Barry Mazur and Mark Schiefsky and the other students of "Geometry and Mechanics" at Harvard University, Spring of 2011-2012.

However, I have argued the terms of that debate *mean different things* for different people. In the wake of efforts to automate proof, both formality and intuition have new meanings. Computers didn't just prompt new interest and debate about intuition and formalization. Engagement with computers produced quite new meanings of those terms.

In the three programs discussed in this dissertation, for example, we have seen different formulations of intuition - as heuristic search, as "Eureka" moments, as a "weighting mechanism." We have also seen intuition *redirected.* In the case of the AURA, input human intuitions tended to emerge from empirical experience with the program itself, rather than from reflection on the mathematical problem at hand. In his imaginations of a new discipline - inferential analysis - Hao Wang thought human insights should be directed away from specific problems and toward general problem-solving methods, toward the "algorithmic dimension" of mathematical problems. The character of intuition and its place in problem solving developed differently as communities fashioned a place for computers in proof.

So too are the character of form and formalism at stake in the automation of proof. As we saw, one basic building block of logical-deductive systems - inference rules - were treated and fashioned quite differently in each of these programs. We saw inference manifested as list processing operations, as eleven rules for "pulling apart" logical propositions according to Herbrand's Theorem, as a method of pattern recognition in sequential tables, and as as the machine-oriented Resolution Principle. Each of these computer operations as a transformation of the basic infrastructure of the logic presented in *Principia Mathematica*.

These were different processes for moving around within formal systems, for preserving truth through transformation. Each was produced to accommodate the affordances of modern digital computers. Each was a translation into the material and

formal languages of computing machines. Each was what I have called a *reformalism.*

But crucially, these transformations of intuition, of inference, of formalism can only be seen if you look at *implementation*. At high enough levels of abstraction, these differences disappear. Newell-Simon-Shaw's intuition is just like Polya's. Wos' intuition is just like Archimedes. The character of intuition and of formality are precisely what's at stake in the automation of proof. Where does intuition fit in the work proof? To explore that question, we must attend to the constantly shifting character and orientation of intuition itself.

Sources

Archive Collections and Abbreviations

AMT	Alan Turing Archive Collection. King's College, Cambridge University. Cambridge, U.K.
AAR-FMR	Association of Automated Reasoning Financial and Membership Records, 1983 - 2000.
CMU-AN	Carnegie Mellon University Archives, Allen Newell Collection. Pittsburgh, PA.
CMU-HS	Carnegie Mellon University Archives, Herbert Simon Collection. Pittsburgh, PA.
CMU-PM	Carnegie Mellon University Archives, Pamela McCorduck Collection. Pittsburgh, PA.
FOI/PA#1193350-0	Federal Bureau of Investigation. Freedom of Information Act #1193350-0. Hao Wang File.
HU-BD	Harvard University Archive Collection, Burton Dreben Collection. Cambridge, MA.
HU-CL	Harvard University Archive Collection, Records of the Computation Laboratory, 1944 - 1961.
HU-HA	Harvard University Archive Collection, Papers of Howard A. Aiken, 1900 - 1973.
HU-HB	Harvard University Archive Collection, Papers of Harvey Brooks, 1930s- 1980s.
MIT-AC-268	Massachusetts Institute of Technology, Laboratory for Computer Science (Research Records 1961 - 1985). Cambridge, MA.
MIT-AC-282	Massachusetts Institute of Technology, Laboratory for Computer Science (Research Records 1961 - 1999). Cambridge, MA.
MU-BR	McMaster University Archives, Bertrand Russell Collection. Hamilton, ON.
NMAH.AC.0508	Archives Center, National Museum of American History, John Clifford Shaw Papers. Washington, D.C.
NIU-UA10	Northern Illinois University, Academic Computing Services.

RAC-RF	Rockefeller Archive Center, Rockefeller Foundation Collection. Sleepy Hollow, NY.
RAC-RU-HW	Rockefeller Archive Center, Rockefeller University Collection, Hao Wang Papers. Sleepy Hollow, NY.
UTA-CAH-WB	University of Texas at Austin, Briscoe Center for American History, Woodrow "Woody" Bledsoe Papers, 1950 - 1996. Austin, TX.

Oral Histories and Personal Correspondence

Andrews, Peter. In person interview, Manchester, UK. 28 June 2012.

Davis, Martin. In person interview, Manchester UK. 27 June 2012.

Friedman, Joyce. In person interview, Cambridge, MA. 13 April 2012.

Floyd, Juliet. In person interview, Cambridge, MA. 11 September, 2012.

Gelernter, Herbert. In person interview, Stonybrook, NY. 19 September 2011.

Loveland, Donald. Email to author. 30 June, 2010.

Moses, Joel. In person interview, Cambridge, MA. 10 September, 2012.

Noftsker, Russell. In person interview, Cambridge, MA. 29 April 2013.

Shankar, Natarajan. In person interview, Manchester UK. 29 June 2012.

Overbeek, Ross. Phone interview. 11 November 2010.

———. Email to author. 11 November 2010.

Paulson, Lawrence. In person interview, Manchester UK. 28 June 2012.

Rabin, Michael. In person interview, Cambridge, MA. 4 December 2013.

Sutcliffe, Geoff. In person interview, Manchester, UK. 27 June 2012.

Voronkov, Andrei. In person interview, Manchester, UK. 26 June 2012.

Wos, Lawrence. Phone interview. 4 November 2010.

Technical Reports

IBM Corporation, *IBM System/360- Principles of Operation*. New York, NY: IBM Publications, 1964.

Newell, Allen. "The Capacity of a Railroad Freight Yard (A Survey of the Problem - Not the Solution)." RAND Document No. RM-555. 14 June 1950.

———. "Description of the Air-Defense Experiments II. The Task Environment." RAND Publication No. P-659. 17 October, 1955.

Newell, Allen; Herbert Simon. "The Logic Theory Machine: A Complex Information Processing System." RAND Publication P-868. 12 July 1956.

————————. "The Processes of Creative Thinking." RAND Technical Report No. P-1320. 16 May 1958.

Newell, Allen; John Clifford Shaw. "Programming the Logic Theory Machine." RAND Publication P-954. February 1957.

Ng, Edward. "Symbolic-Numeric Interface: A Review." NASA Jet-Propulsion Lab, Technical Report N80-16768. 1 January 1980.

Rabin, Michael O. "Degree of Difficulty of Computing a Function and Hierarchy of Recursive Sets." Technical Report No. 2, Office of Naval Research. 1960.

Smith, Brian. *Reference Manual for the Environmental Theorem Prover: An Incarnation of AURA (Automated Reasoning Assistant).* Argonne National Laboratory Technical Report ANL-88-2. March 1988.

Simon, Herbert. "Observations and Comments on the Organization Studies of the Systems Research Laboratory."RAND Publication No. RM-922. 1952.

Veroff, Robert. "Canonization and Demodulation." Argonne National Laboratory Technical Report ANL-81-6. February 1981.

von Neumann, John. "First Draft of a Report on the EDVAC" Contract No. W-670-ORD-4962 between the U.S. Army Ordnance Department and the University of Pennsylvania. June 30, 1945 [transcribed and edited by Michael D. Godfrey, 1992].

Wang, Hao. "The Control of Errors." Burroughs Corporation Technical Memorandum No. 54-59, Project No. R5781.

Ware, Willis. "Johnniac Eulogy." RAND Document P-3313. March 1966.

Published Primary Sources

Albers, Donald. "Polite Applause for a Proof of One of the Great Conjectures of Mathematics: What is a Proof Today?" in *The Two-Year College Mathematics Journal*, Vol. 12, No. 2 (1981): 82.

Allen, J.; D. Luckham. "An Interactive Theorem-Proving Program" in *Automation of Reasoning 2: Classical Papers on Computational Logic 1967 - 1970*, eds. Jörg Siekmann, Graham Wrightson. Berlin, Germany: Springer, 1983: 417 - 434.

Appel, Kenneth. "The Nature of Proof: Limits and Opportunities" in *The Two-Year College Mathematics Journal*, Vol. 12, No. 2 (1981): 118 - 119.

Appel, Kenneth; Wolfgang Haken. "Every Planar Map is Four Colorable" in *Illinois Journal of Mathematics*, Vol. 21, No. 3 (1997): 429 - 490.

———. "The Four Color Proof Suffices" in *The Two-Year College Mathematics Journal*, Vol. 12, No. 1 (1986): 10 - 20.

Bell, Gordon; Allen Newell. *Computer Structures; Readings and Examples.* New York, NY: McGraw-Hill, 1971.

Benzmueller, Christoph; Chad Brown; Joerg Siekmann, eds. *Reasoning in Simple Type Theory: Festschrift in Honor of Peter B. Andrews on his 70th Birthday* [*Studies in Logic* Vol. 17]. New York, NY: College Publications, 2008.

Bledsoe, Woody. "Some Automatic Proofs in Analysis" in *Automated Theorem Proving: After 25 Years*, ed. Donald Loveland. Providence, RI: American Mathematical Society, 1983: 89 - 118.

Boole, George. *An Investigation of the Laws of Thought on Which are Founded the Mathematical Theories of Logic and Probabilities.* London, U.K.: Walton & Maberly, 1984.

Borwein, Jonathan; David Bailey. *Mathematics by Experiment, 2nd Edition: Plausible Reasoning in the 21st Century.* New York, NY: A K Peters, 2008.

Brillinger, Peter; Doran Cohen. *Introduction to data structures and non-numeric computation.* Upper Saddle River, NJ: Prentice-Hall Series in Automatic Computation, 1972.

Chapin, Ned. *Programming in Assembly Language*, second edition. New York, NY: McGraw Hill, 1973 [1968].

Church, Alonzo. "An Unsolvable Problem in Number Theory" in *American Journal of Mathematics*, Vol. 58, No. 2 (1936): 345 - 363.

———. "A Note on the Entscheidungsproblem" in *Journal of Symbolic Logic*, Vol. 58 (1936): 40 - 41.

Davis, Martin. *Computability and Unsolvability.* New York, NY: Dover, 1973 [1958].

———. "A Computer Program for Presburger's Algorithm" in *Automation of Reasoning 1: Classical Papers on Computational Logic 1957 - 1966*, J. Siekmann, G. Wrightson, eds. Berlin, Germany: Springer-Verlag, 1983: 41 - 48.

———. *Engines of Logic: Mathematicians and the Origin of the Computer.* New York, NY: Norton, 2001.

———. "Hao Wang's Contributions to Mechanized Deduction and to the Entscheidungsproblem" in *Hao Wang: Logician and Philosopher*

[*Texts in Philosophy* 16], eds. Charles Parson, Montgomery Link. London, U.K.: College Publications, 2011: 73 - 78.

Davis, Martin; Hilary Putnam. "A Computing Procedure for Quantification Theory" in *Journal of the Association for Computing Machinery*, Vol. 7, No. 3 (1960): 201 - 215.

Eckert, Wallace; Rebecca Jones. *Faster, faster: A Simple Description of a Giant Electronic Calculator and the Problems it Solves.* New York, NY: McGraw-Hill, 1955.

Feigenbaum, Edward; Julian Feldman, eds. *Computers and Thought.* New York, NY: McGraw Hill, 1963.

Gardner, Martin. *Logic Machines and Diagrams.* Chicago, IL: University of Chicago Press, 1983.

Gelernter, Herbert. "Realization of a Geometry-Theorem Proving Machine" in *Computers and Thought*, eds. Edward Feigenbaum, Julian Feldman. Cambridge, MA: The MIT Press, 1995: 134 - 152.

Gelernter, Herbert; J.R. Hansen; C.L. Gerberich. "A FORTRAN-Compiled List-Processing Language" in *Journal of the Association of Computing Machinery*, Vol. 7, No. 2 (1960).

Gentzen, Gerhard. "Untersuchungen über das Logische Schliessen," in *Mathematik Zeitschrift*, Vol. 39 (1934-1935): 176 - 210.

Gödel, Kurt. "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter System I" in *Monatshefte für Mathematik,* Vol. 38, No. 1 (1931): 173 - 198.

Goldstine, Herman; Adele Goldstine. "The Electronic Integrator and Computer (ENIAC)" in *Mathematical Tables and Other Aids to Computation*, Vol. 2, No. 15 (1946): 97 - 110.

Guard, J.R.; F.C. Aglesby; J.H. Bennett; L.G. Settle. "Semi-Automated Mathematics" in *Automation of Reasoning 2: Classical Papers on Computational Logic 1967 - 1970*, eds. Jörg Siekmann, Graham Wrightson. Berlin, Germany: Springer, 1983: 203 - 216.

Hadamard, Jacques. *The Psychology of Invention in the Mathematical Field.* Princeton, NJ: Princeton University Press, 1945.

Hall, Patrick. *Computational Structures: An Introduction to Non-Numeric Computing.* London, U.K.: MacDonald and Janes, 1975.

Hartree, Douglas. "The ENIAC, An Electronic Computing Machine" in *Nature*, Vol. 158 (1946): 500 - 506.

Heijenoort, Jean van. *Frege and Gödel: Two Fundamental Texts in Mathematical Logic.* Lincoln, NW: iUniverse, 1999.

———————. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879 - 1931.* Cambridge, MA: Harvard University Press, 2002.

Hilbert, David. "Mathematical Problems" in *Bulletin of the American Mathematical Society*, Vol. 8, No. 10 (1902): 437 - 479.

Hilbert, David; William Ackermann. *Grundzüge Der Theoretischen Logik.* Berlin, Germany: Springer Verlan, 1928.

Hilbert David; Wilhelm Ackermann. *Principles of Mathematical Logic*, trans. L. Hammond, G. Keckie, R. Steinhardt. New York, NY: Chelsea Publishing Company, 1950 [1928].

Hilbert, Martin; Priscilla Lopez. "The World's Technological Capacity to Store, Communicate, and Compute Information" in *Science* Vol. 332, No. 60 (April 2011): 60 - 65.

Kleene, Stephen. *Mathematical Logic.* New York, NY: Dover, 1967.

Knuth, Donald. *Art of Computing Programming, Volume 4: Generating All Trees - History of Combinatorial Generation.* New York, NY: Addison-Wesley Professional, 2006.

Loveland, Donald. "Automated Theorem-Proving: A Quarter Century-Review" in *Automated Theorem-Proving: After 25 Years* [*Contemporary Mathematics*, Vol. 29]. Providence, RI: The American Mathematical Society, 1983: 1 - 46.

———————. *Automated Theorem-Proving: After 25 Years* [*Contemporary Mathematics*, Vol. 29]. Providence, RI: The American Mathematical Society, 1983.

McCharen, J.; Ross Overbeek; Lawrence Wos. "Problems and Experiments for and with Automated Theorem-Proving Programs" in *IEEE Transactions on Computers*, C-25, No. 8 (August 1976): 773 - 782.

McCune, William. "Solution of the Robbins Problem" in *Journal of Automated Reasoning*, Vol. 19, No. 3 (1997): 263 - 276.

Moses, Joel; Michael Dertouzous. *The Computer Age: A Twenty-Year View.* Cambridge, MA: The MIT Press, 1980.

Newell, Allen. "The Chess Machine: An Example of Dealing with a Complex Task by Adaptation" in *Proceedings of the 1955 Western Joint Computer Conference.* 1 March, 1955: 101 - 108.

Newell, Allen; Herbert Simon, John Clifford Shaw. "Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristics" in *Proceedings of the Joint Western Computer Conference* (1957): 218 - 130.

Newell, Allen; Herbert Simon. *Human Problem Solving.* Vol. 104, No. 8. Englewood Cliffs, NJ: Prentice-Hall, 1972.

Ng, Edward. "Introduction" in *Proceedings of the Symposium on Symbolic and Algebraic Computation.* Berlin, German: Springer, 1979: 2.

Overbeek, Ross; Ewing Lusk, "Wos and Automated Deduction at ANL: The Ethos" in *Automated Reasoning and its Applications: Essays in Honor of Larry Wos.* Cambridge, MA: The MIT Press, 1997: 1 - 12.

Paulson, Lawrence. "The Foundation of a Generic Theorem Prover" in *Journal of Automated Reasoning*, Vol. 5, No. 3 (September 1989): 363 - 397.

———. *The Isabelle Reference Manual*, July 2008: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.9052&rep=rep1&type=pdf`.

Peano, Giuseppe. *Formulaire de Mathématique* (*Rivisita di matematica*), 1885.

———. "The principles of arithmetic, presented by a new method" in *A Source Book in Mathematical Logic, 1879 - 1931*, ed. Jean van Heijenoort. Cambridge, MA: Harvard University Press, 1967: 83 - 97.

Polya, George. *How to Solve It: A New Aspect of Mathematical Method.* Princeton, NJ: Princeton University Press, 1945.

———. *Induction and Analogy in Mathematics*, Vol. I of *Mathematics and Plausible Reasoning.* Princeton, NJ: Princeton University Press, 1954.

———. *Patterns of Plausible Inference*, Vol. II of *Mathematics and Plausible Reasoning.* Princeton, NJ: Princeton University Press, 1954.

———. *Mathematical Discovery: On Understanding, Learning and Teaching Problem Solving*, Vols. 1, 2. Hoboken, NJ: John Wiley & Sons, 1965.

———. *Mathematical Methods in Science.* Washington, D.C.: The Mathematical Association of America, 1977.

Robinson, John Alan. "A Machine-Oriented Logic Based on the Resolution Principle" in *Journal of the Association for Computing Machinery*, Vol. 12, No. 1 (January 1965): 23 - 41.

———. "Logic and Logic Programming" in *Communications of the Association for Computing Machinery*, Vol. 35, No. 3 (March 1982): 41 - 65.

———. "Davis, Putnam, UNIVAC, FORTRAN, and other miracles: How I got involved with Computational Logic and What Happened Next" at *International Joint Conference on Automated Reasoning,* CADE Invited Speaker. 29 June, 2012.

Robinson, John Alan; Andrei Voronkov eds. *Handbook of Automated Reasoning Vols. 1, 2.* Cambridge, MA: The MIT Press, 2001.

Russell, Bertrand. *My Philosophical Development.* London, U.K.: Allen and Unwin, 1959.

Selfridge, Oliver; G.P. Dineen. "Pattern Recognition and Modern Computers" in *Proceedings of the 1955 Western Joint Computer Conference.* 1 March 1955: 91 - 93.

Shannon, Claude. "A Mathematical Theory of Communication" in *Bell Systems Technical Journal*, Vol. 27 (October 1948): 379 - 423.

Shannon, Claude; John McCarthy. *Automata Studies.* Princeton, NJ: Princeton University Press, 1956.

Siekman, J.; G. Wrightson, eds. *The Automation of Reasoning: Classical Papers on Computational Logic, Volumes I and II.* Berlin, Germany: Springer, 1983.

Simon, Herbert. *The New Sciences of Management Decision.* New York, NY: Harper & Row, 1960.

———. *Models of My Life.* New York, NY: Basic Books, 1991.

Styazhkin, N. I. *History of Mathematical Logic from Leibniz to Peano.* Cambridge, MA: The MIT Press, 1969.

Suttcliffe, Geoff; Christian Suttner; Theodor Yemenis. "The TPTP Problem Library" in *Proceedings of the 12th Conference in Automated Deduction* (1994): 252 - 266.

Turing, Alan. "On Computable Numbers with an Application to the *Entscheidungsproblem*" in *Proceedings of the London Mathematical Society*, Vol. 42 (1937): 230 - 265.

———. "Computing Machinery and Intelligence" in *Mind*, Vol. 19 (1950): 433 - 460.

———. *The Essential Turing: Seminal Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life Plus the Secrets of Enigma*, ed. Jack Copeland. Oxford, U.K.: Oxford University Press, 2004.

van Emden. Interview with John Alan Robinson in *Association for Automated Reasoning Newsletter*, No. 89 (October 2010). Online edition.

Veroff, Robert, ed. *Automated Reasoning and Its Applications: Essays in Honor of Larry Wos.* Cambridge, MA: The MIT Press, 1993.

Wang, Hao. "An Economical Ontology for Classical Arithmetic." PhD Dissertation submitted to the Department of Philosophy, Harvard University, 1948. *Harvard University Archive Collection*, HU 90.5483.2.

———. "A Variant to Turing's Theory of Computing Machines" in *Journal of the Association for Computing Machinery*, Vol. 4, No. 1 (1957): 63 - 92.

———. "Toward Mechanical Mathematics" in *IBM Journal of Research and Development*, Vol. 4, No. 1 (1960): 2 - 22.

———. "Proving Theorems by Pattern Recognition I" in *Communications of the ACM*, Vol. 3, No. 4 (1960): 220 - 234.

———. "Proving Theorems by Pattern Recognition II" in *The Bell System Technical Journal,* Vol. 40, No. 1 (January 1960): 1 - 42.

———. *Popular Lectures on Mathematical Logic.* New York, NY: Litton Educational Publishing, 1981.

———. "Computer Theorem Proving and Artificial Intelligence" in *Automated Theorem-Proving: After 25 Years*, ed. Donald Loveland. Providence, RI: American Mathematical Society, 1983: 49 - 70.

———. *Reflections on Kurt Gödel.* Cambridge, MA: The MIT Press, 1987.

———. *A Logical Journey: From Gödel to Philosophy.* Cambridge, MA: The MIT Press, 1996.

———. *Beyond Analytic Philosophy: Doing Justice to What We Know.* Cambridge, MA: The MIT Press, 1996 [1988].

———. "From Kunming to New York," trans. Richard Jandovitz, Montgomery Link in *Hao Wang: Logician and Philosopher* [*Texts in Philosophy* 16]. London, U.K.: College Publications, 2011: 39 - 48.

Weyl, Herman. "Über die neue Grundlagenkrise der Mathematik" in *Mathematische Zeitschrift*, Vol. 10 (1921): 39 - 79.

Whitehead, Alfred North; Bertrand Russell. *Principia Mathematical*, Vols. I - III. Cambridge, U.K.: Cambridge University Press, 1910, 1912, 1913.

Wiener, Norbert. *Cybernetics or Control and Communication in the Animal and the Machine.* Cambridge, MA: The MIT Press, 1948.

Winker, Steven. "Generation and Verification of Finite Models and Counter-Examples Using an Automated Theorem Prover Answering Two Open Questions" in *Journal of the Association for Computing Machinery*, Vol. 29 (1982): 273 - 284.

Winker, Steven; Lawrence Wos. "Automated Generation of Models and Counterexamples and its Application to Open Questions in Ternary Boolean Algebra" in *Proceedings of the Eigth International Symposium on Multiple-Valued Logic.* Rosemont, IL: *IEEE* and *ACM Publications*, 1978: 251 - 256.

Winker, Steven; Lawrence Wos; Ewing Lusk. "Semigroups, antiautomorphisms, and Involutions: A Computer Solution to an Open Problem, I" in *Mathematics of Computation*, Vol. 37 (1981): 533 - 545.

Wos, Lawrence. "The Unit Preference Strategy in Theorem Proving" in *Collected Works of Larry Wos*, Vol. 1. Hackensack, NJ: World Scientific Publishing Company, 2001: 17 - 28 [originally *American Federation of Information Processing Society*, Proceedings 26 (1964): 615 - 621].

————. "Solving Open Questions with an Automated Theorem-Proving Program" in *Proceedings of the Conference on Automated Deduction* (1982): 1 - 31.

————. "A Journal is Born" in *Journal of Automated Reasoning*, Vol. 1, No. 1 (1985): 1 - 3.

Wos, Lawrence; Stephen Winker. "Open Questions Solved with the Assistance of AURA" in *Automated Theorem-Proving: After 25 Years* [Vol. 29 of *Contemporary Mathematics*], eds. Woody Bledsoe, Donald Loveland. Providence, RI: American Mathematical Society, 1984: 73 - 88.

Wos, Lawrence; Steven Winker; Ewing Lusk. "An Automated Reasoning System" in *AFIPS Conference Proceedings*, Vol. 50 (1981): 697 - 702.

Secondary Sources

Aczel, Amir. *A Strange Wilderness: The Lives of the Great Mathematicians*. New York, NY: Sterling, 2011.

Adam, Alison. *Artificial Knowing: Gender and the Thinking Machine*. New York, NY: Routledge, 1998.

Agar, John. "Histories of Computing: Approaches, New Directions" in *History and Technology*, Vol. 15 (1998): 1 - 5.

————. "What Difference Did Computers Make?" in *Social Studies of Science*, Vol. 36, No. 6 (2006): 869 - 907.

Ahmed, Sara. *Differences that Matter: Feminist Theory and Postmodernism*. Cambridge UK: Cambridge University Press, 1998.

Alexander, Amir. *Geometrical Landscapes: The Voyagers of Discovery and the Transformation of Mathematical Practice*. Stanford, CA: Stanford University Press, 2002.

————. *Dual at Dawn: Heroes, Martyrs, and the Rise of Modern Mathematics*. Cambridge, MA: Harvard University Press, 2010.

————. "From Voyagers to Martyrs: Toward a Storied History of Mathematics" in *Circles Disturbed: The Interplay of Mathematics and Narrative*, eds. Apostolos Doxiadis, Barry mazur. Princeton, NJ: Princeton University Press, 2012: 1 - 51.

Alexanderson, Gerald. *The Random Walks of George Polya.* Washington, D.C.: Mathematical Association of America, 2000.

Ascher, Marcia. *Mathematics Elsewhere: An Exploration of Ideas Across Cultures.* Princeton, NJ: Princeton University Press, 2004.

Ashworth, William. "The Calculating Eye: Baily, Herschel, Babbage and the Business of Astronomy" in *British Journal for the History of Science*, Vol. 27 (1994): 409 - 441.

————————. "Memory, Efficiency, and Symbolic Analysis: Charles Babbage, John Herschel, and the Industrial Mind" in *Isis*, Vol. 87 (1996): 629 - 653.

Aspray, William. "The Scientific Conceptualization of Information: A Survey" in *Annals of the History of Computing*, Vol. 7, No. 2 (1985): 117 - 140.

————————. "Was Early Entry a Competitive Advantage? US Universities that Entered Computing in the 1940s" in *IEEE Annals of the History of Computing*, Vol. 22, No. 3 (July-September 2000): 42 - 87.

————————. "The Institute for Advanced Study Computer: A Case Study in the Application of Concepts from the History of Technology" in *The First Computers: history and Architecture*, ed. Raúl Rojas, Ulf Hashagen. Cambridge, MA: The MIT Press, 2002: 179 - 194.

Bashe, Charles; Lyle Johnson; John Palmer; Emerson Pugh. *IBM's Early Computers: A Technical History.* Cambridge, MA: The MIT PRess, 1986.

Belhoste, Bruno. "The École Polytechnique and Mathematics in Nineteenth-Century France" in *Changing Images in Mathematics: From the French Revolution to the New Millennium*, ed. Umberto Bottazini, Amy Dahan Dalmedico. New York, NY: Routledge, 2001.

Beniger, James. *The Control Revolution: Technological and Economic Origins of the Information Society.* Cambridge, MA: Harvard University Press, 1986.

Biagioli, Mario; Peter Jaszi; Martha Woodmansee, eds. *Making and Unmaking Intellectual Property: Creative Production in Legal and Cultural Perspective.* Chicago, IL: University of Chicago Press, 2011.

Bijker, Wiebe. *Of Bicycles, Bakelites, and Bulbs: Toward a Theory of Sociotechnical Change.* Cambridge, MA: The MIT Press, 1995.

Bijker, Wiebe; John Law, ed. *Shaping Technology/ Building Society: Studies in Sociotechnical Change.* Cambridge, MA: The MIT Press, 1994.

Bloor, David. "Wittgenstein and Hannheim on the Sociology of Mathematics" in *Studies in History and Philosophy of Biology and Biomedical Sciences*, Vol. 4, No. 2 (1973): 173 - 191.

————————. "Polyhedra and the Abominations of Leviticus: Cognitive Styles in Mathematics" in *The British Journal for the History of Science*, Vol. 11, No. 3 (1978): 245 - 272.

————————. "The Living Foundations of Mathematics" in *Social Studies of Science*, Vol. 17, No. 2 (1987): 337 - 358.

————————. "What Can the Sociologist of Knowledge Say About 2 + 2 = 4" in *Mathematics, Education, and Philosophy: An International Perspective*, ed. Paul Ernest. London, U.K.: The Falmer Press, 1994: 21 - 32.

Bolter, Jay; Richard Grusin. *Remediation: Understanding New Media.* Cambridge, MA: The MIT Press, 1999.

Bolter, Jay. *Writing Spaces: Computers, Hypertext, and the Remediation of Print.* New York, NY: Routledge, 2001.

Bos, Henk; Herbert Mehrtens. "The Interaction of Mathematics and Society in History and Some Exploratory Remarks" in *Historia Mathematica*, Vol. 4, No. 1 (1977): 7 - 30.

Bos, Henk. "The Structure of Descartes's *Géométrie*" in *Lectures in the History of Mathematics.* Berlin, Germany: Springer, 1993: 37 - 58.

Bottazini, Bruno; Dalmedico, Amy Dahan, eds. *Changing Images in Mathematics: From the French Revolution to the New Millennium.* New York, NY: 2001.

Bourdieu, Pierre. *Outline of a Theory of Practice*, trans. Richard Nice. Cambridge, U,K.: Cambridge University Press, 1977.

————————. *The Logic of Practice.* Stanford, CA: Stanford University Press, 1990.

Burks, Arthur; Alice Burks. "The ENIAC: First General-Purpose Electronic Computer" in *Annals of the History of Computing*, Vol. 3, No. 4 (October 1981): 310 - 389.

Cajori, Florian. *A History of Mathematical Notations.* New York, NY: Dover, 2011.

Calinger, Ronald. "The Mathematics Seminar at the University of Berlin: Origins, Founding, and the Kummer-Weierstrass Years" in *Vita Mathematica: Historical Research and Integration with Teaching*, ed. Ronald Calinger. Cambridge, U.K.: Cambridge University Press, 1996.

Campbell-Kelly, Martin. *Computer: A History of the Information Machine*. New York, NY: Basic Books, 1997.

Cantwell Smith, Brian. *On the Origin of Objects*. Cambridge, MA: The MIT Press, 1998.

Cardiadiego, Alejandro. "Rewriting the History of the Foundations of Mathematics at the Turing of the Century" in *Historia Mathematica*, Vol. 13 (1986): 39 - 41.

Casper, Monica. "Reframing and grounding nonhuman agency: What makes a fetus an agent" in *American Behavioral Scientist,* Vol. 37 (1994): 839 - 856.

Ceruzzi, Paul. *A History of Modern Computing.* Cambridge, MA: The MIT Press, 2003 [1998].

Chun, Wendy. *Programmed Visions: Software and Memory.* Cambridge, MA: The MIT Press, 2011.

Clark, Andy; David Chalmers. "The Extended Mind" in *Analysis*, Vol. 58, No. 1 (1998): 7 - 19.

Clark, Andy. *Supersizing the Mind: Embodiment, Action, and Cognitive Extension.* Oxford, U.K.: Oxford University Press, 2008.

Collins, Harry. *Artificial Experts: Social Knowledge and Intelligent Machines.* Cambridge, MA: The MIT Press, 1992.

Collins, Martin. *Planning for Modern War: RAND and the Air Force 1945 - 1950.* PhD Dissertation submitted to the Department of History, University of Maryland, College Park, MD: 1998.

Copeland, Jack. *Turing: Pioneer of the Information Age.* Oxford, U.K.: Oxford University Press, 2012,

Cordeschi, Roberto. *The Discovery of the Artificial: Behavior, Mind and Machines Before and Beyond Cybernetics.* Berlin, Germany: Springer, 2002.

Corfield, David. *Towards a Philosophy of Real Mathematics.* Cambridge, U.K.: Cambridge University Press, 2006.

Corry, Leo. "The Origins of Eternal Truth in Modern Mathematics: Hilbert to Bourbaki and Beyond" in *Science in Context*, Vol. 12 (1998): 137 - 183.

————————. "Axiomatics, Empiricism, and Anschauung in HIlbert's Conception of Geometry: Between Arithmetic and General Relativity" in *The Architecture of Modern Mathematics: Essays in History and Philosophy*, ed. José Dominguez, Jeremy Gray. Oxford, U.K.: Oxford University Press, 2006.

Crowther-Heyck, Hunter. *Herbert A. Simon: The Bounds of Reason in Modern America*. Baltimore, ML: Johns Hopkins University Press, 2005.

————————. "Defining the Computer: Herbert Simon and the Bureaucratic Mind - Part I" in *IEEE Annals of the History of Computing*, Vol. 30, No. 2 (2008): 42 - 51.

————————. "Defining the Computer: Herbert Simon and the Bureaucratic Mind - Part II" in *IEEE Annals of the History of Computing*, Vol. 30, No. 2 (2008): 52 - 63.

Csiszar, Alex. "Poincaré and Peano: Bibliographer vs. Natural Philosopher" [unpublished excerpt.]

Damerow, Peter. *Abstraction and Representation: Essays on the Cultural Evolution of Thinking*. Berlin, Germany: Springer, 1996.

Daston, Lorraine. "The Physicalist Tradition in Early Nineteenth Century French Geometry" in *Studies in History and Philosophy of Science*, Vol. 17, No. 3 (1986): 269 - 295.

————————. *Classical Probability in the Enlightenment*. Princeton, NJ: Princeton University Press, 1999.

————————. "Enlightenment Calculations" in *Critical Inquiry*, Vol. 21, No. 1 (Autumn, 1994): 182 - 202.

Dauben, Joseph. *Georg Cantor: His Mathematics and Philosophy of the Infinite*. Princeton, NJ: Princeton University Press, 1990.

De Saussure, Ferdinand. *Course in General Linguistics*. New York, NY: McGraw-Hill, 1959.

Derrida, Jacques. "Structure, Sign, and Play in the Discourse of the Human Science" in *Writing and Difference*. Chicago, IL: University of Chicago Press, 1978: 278 - 293.

————————. *Paper Machine*, trans. Rachel Bowlby. Stanford, CA: Stanford University Press, 2005.

Detlefsen, Michael; Mark Luker. "The Four Color Theorem and Mathematical Proof" in *The Journal of Philosophy*, Vol. 77, No. 12 (1980): 803 - 820.

Dick, Stephanie. "AfterMath: The Work of Proof in the Age of Human-Machine Collaboration" in *Isis*, Vol. 102, No. 3 (September 2011): 494 - 505.

————————. "Machines Who Write" in *IEEE Annals for the History of Computing*, Vol. 35, No. 2 (April-June 2013): 85 - 87.

Doxiadis, Apostolos; Christos Papadimitriou. *Logicomix: An Epic Search for Truth.* London, U.K.: Bloomsbury, 2009.

Doxiadis, Apostolos; Barry Mazur, eds. *Circles Disturbed: The Interplay of Mathematics and Narrative.* Princeton, NJ: Princeton University Press, 2012.

Dreyfus, Hubert. *What Computers Can't Do: The Limits of Artificial Intelligence.* New York, NY: HarperCollins, 1978.

———. *Mind Over Machine: The Power of Human Intuition and Expertise in the Era of the Computer.* Oxford, U.K.: Basil Blackwell, 1986.

———. *What Computers Still Can't Do: A Critique of Artificial Reason.* Cambridge, MA: The MIT Press, 1992.

Edwards, Paul. *The Closed World: Computers and the Politics of Discourse in Cold War America.* Cambridge, MA: The MIT Press, 1996.

———. *A Vast Machine: Computer Models, Climate Data, and the Politics of Global Warming.* Cambridge, MA: The MIT Press, 2013.

Eglash, Ron. *African Fractals: Modern Computing and Indigenous Design.* New York, NY: Rutgers, 1999.

———. "Broken Metaphor: The Master-Slave Analogy in Technical Literature" in *Technology and Culture*, Vol. 48, No. 2 (2007): 360 - 369.

Ensmenger, Nathan. *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise.* Cambridge, MA: The MIT Press, 2010.

———. "The Digital Construction of Technology: Rethinking the History of Computers in Society" in *Technology and Culture*, Vol. 53 (2012): 753 - 776.

———. "Is Chess the Drosophila of Artificial Intelligence? A Social History of an Algorithm" in *Social Studies of Science*, Vol. 42, No. 1 (2012): 5 - 30.

Erickson, Paul. "Mathematical models, rational choice, and the search for Cold War Culture" in *Isis*, Vol. 101 (2010); 386 - 392.

Erickson, Paul; Judy Klein; Lorraine Daston; Rebecca Lemov; Thomas Sturm; Michael Gordin. *How Reason Almost Lost Its Mind: The Strange Career of Cold War* Rationality. Chicago, IL: The University of Chicago Press, 2013.

Fisher, Charles. "The Death of a Mathematical Theory: A Study in the Sociology of Knowledge" in *Archives for History of Exact Sciences*, Vol. 3 (1966): 137 - 159.

———————. "Some Social Characteristics of Mathematicians and Their Work" in *American Journal of Sociology*, Vol. 78, No. 5 (1973): 1094 - 1118.

Foucault, Michel. *The Order of Things: An Archaeology of the Human Science*. New York, NY: Routledge, 2007.

Friedman, Ted. "Filming the "Electronic Brain"" in *Electric Dreams*. New York, NY: NYU Press, 2005: 47 - 78.

Fritsch, Rudolf; Gerda Fritsch. *The Four-Color Theorem: History, Topological Foundations, and Idea of Proof*, trans. Julie Peschke. Berlin, Germany: Springer, 1998.

Fuller, Matthew. *Behind the Blip: Essays on the Culture of Software*. New York, NY: Autonomedia, 2003.

Fuller, Matthew, ed. *Software Studies: A Lexicon*. Cambridge, MA: The MIT Press, 2008.

Galison, Peter. "The Ontology of the Enemy" in *Critical Inquiry*, Vol. 21, No. 1 (1994): 228 - 266.

———————. "Context and Constraints" in *Scientific Practice: Theories and Stories of Doing Physics*, ed. Jed Buchwald. Chicago, IL: University of Chicago Press, 1995: 13 - 41.

———————. *Image and Logic: A Material Culture of Microphysics*. Chicago, IL: University of Chicago Press, 1997.

———————. "Trading Zone: Coordinating Action and Belief" in *The Science Studies Reader*, ed. Mario Biagioli. New York, NY: Routledge, 1999: 137 - 160.

———————. *Einstein's Clocks and Poincaré's Maps: Empires of Time*. New York, NY: Norton, 2004.

Galison, Peter; Bruce Hevly, eds. *Big Science: The Growth of Large Scale Research*. Stanford, CA: Stanford University Press, 1994.

Geertz, Clifford. *The Interpretation of Cultures; Selected Essays*. New York, NY: Basic Books, 1973.

Gillies, Donald, ed. *Revolution in Mathematics*. Oxford, U.K.: Oxford University Press, 1999.

Gitelman, Lisa. *Scripts, Grooves, and Writing Machines: Representing Technology in the Edison Era*. Stanford, CA: Stanford University Press, 2000.

———————. *Paper Knowledge: Toward a Media History of Documents (Sign, Storage, Transmission)*. Durham, NC: Duke University Press, 2014.

Goldfarb, Warren. "Introduction" in *Jacques Herbrand: Logical Writings*, ed. Warren Goldfarb. Dortrecht, Holland: Reidel Publishing, 1971: 1 - 20.

Goody, Jack. *The Domestication of the Savage Mind*. Cambridge, U.K.: Cambridge University Press, 1977: 74 - 111.

————————. "Writing and Formal Operations" in *Africa: Journal of the International African Institute*, Vol. 47, No. 3 (1977): 289 - 304.

Grabiner, Judith. "Is Mathematical Truth Time-Dependent?" in *American Mathematical Monthly*, Vol. 81 (1974): 354 - 365.

————————. "Computers and the Nature of Man: A Historian's Perspective on Controversies About Artificial Intelligence" in *Bulletin of the American Mathematical Society*, Vol. 15 (1986): 113 - 126.

Graham, Loren; Jean-Michel Kantor. *Naming Infinity: A True Story of Religious Mysticism and Mathematical Creativity*. Cambridge, MA: Belknap Press, 2009.

Grattan-Guiness, Ivor. *The Search for Mathematical Roots, 1870 - 1940: Logics, Set Theories and the Foundations of Mathematics from Cantor through Russell to Gödel*. Princeton, NJ: Princeton University Press, 2002.

————————. "Algebras, Projective Geometry, Mathematical Logic, and Constructing the World: Intersections in the Philosophy of Mathematics of A.N. Whitehead" in *Historia Mathematical*, Vol 29 (2002): 427 - 462.

Gray, Chris. *Cyborg Handbook*. New York, NY: Routledge, 1995.

Gray, Jeremy. "Anxiety and Abstraction in Nineteenth-Century Mathematics" in *Science in Context*, Vol. 17 (2004): 23 - 47.

Grier, David. "The ENIAC, the verb "to program" and the emergence of the digital computer" in *IEEE Annals of the History of Computing*, Vol. 18, No. 1 (1996): 51 - 55.

Grier, John Alan. *When Computers Were Human*. Princeton, NJ: Princeton University Press, 2005.

Gruenberger, F. J. "The History of the JOHNNIAC" in *Annals of the History of Computing*, Vol. 1, No. 1 (1979): 49 - 64.

Hacking, Ian. "What is logic?" in *The Journal of Philosophy*, Vol. 76, No. 6 (June 1979): 285 - 319.

————————. *Historical Ontology*. Cambridge, MA: Harvard University Press, 2004.

———————. "Trees of Logic, Trees of Porphyry" in *Advancements of Learning: Essays in Honor of Palo Rosse* [Vol. 62 of *Biblioteca di Nuncis*]. Verona, Italy: L.S. Olschki, 2007: 219 - 261.

Haigh, Thomas. "Inventing Information Systems: The Systems Men and the Computer, 1950 - 1968" in *The Business History Review*, Vol. 75, No. 1 (Spring 2001): 15 - 61.

———————. "Software in the 1960s as Concept, Service, and Product" in *IEEE Annals of the History of Computing*, Vol. 24, No. 1 (2002): 5 - 13.

———————. "How Data Got its Base: Information Storage Software in the 1950s and 1960s" in *IEEE Annals of the History of Computing*, Vol. 31, No. 4 (2009): 6 - 25.

Haigh, Thomas; Mark Priestley; Crispin Rope. "Reconsidering the Stored-Program Concept" in *IEEE Annals of the History of Computing*, Vol. 36, No. 1 (2014): 4 - 17

———————. "Betting on ENIAC: Nuclear Monte Carlo Simulations, 1947-50" in *IEEE Annals of the History of Computing,* forthcoming.

Haraway, Donna. *Simians, Cyborgs, and Women: The Reinvention of Nature*. New York, NY: Routledge, 1991.

Harper, Kristine. *Weather by the Numbers: The Genesis of Modern Meteorology*. Cambridge, MA: The MIT Press, 2012.

Harris, Michael. "Do Androids Prove Theorems in Their Sleep" in *Circles Disturbed: The Interplay of Mathematics and Narrative*, eds. Apostolos Doxiadis, Barry Mazur. Princeton, NJ: Princeton University Press, 2012.

Hartmanis, Juris. "Observations about the Development of Theoretical Computer Science" in *Annals of the History of Computing*, Vol. 3, No. 1 (1981): 42 - 51.

Hayles, Katherine. *Writing Machines*. Cambridge, MA: The MIT Press, 2002.

Helmreich, Stefan. *Silicon Second Nature: Culturing Artificial Life in a Digital World*. Berkeley, CA: University of California Press, 1998.

———————. *Sounding the Limits of Life: Essays in the Anthropology of Biology and Beyond*. Princeton: NJ: Princeton University Press, forthcoming.

Hodges, Andrew. *Alan Turing: The Enigma*. Princeton, NJ: Princeton University Press, 2012 [1983].

Hofstadter, Douglas. *Gödel, Escher, Bach: An Eternal Golden Braid*. New York, NY: Basic Books, 1979.

Holl, Jack. *Argonne National Laboratory, 1946 - 96*. Chicago, IL: University of Chicago Press, 1997.

Hounshell, David. "The Medium is the Message, or How Context Matters: The RAND Corporation Builds an Economics of Innovation, 1946 - 1962" in *Systems, Experts, and Computers: The Systems Approach in Management and Engineering, World War II and After*, eds. Hughes, Hughes. Cambridge, MA: The MIT Press, 2000: 255 - 310.

Hudecek, Jiri. "You Fight Your Way, I Fight My Way: Wu Wen-Tsun and Traditional Chinese Mathematics." Dissertation Submitted to the Department of History and Philosophy of Science, University of Cambridge. November 2011.

Hughes, Agatha; Thomas Hughes, eds. *Systems, Experts, and Computers: The Systems Approach in Management and Engineering, World War II and After.* Cambridge, MA: The MIT Press, 2000.

Husbands, Phil; Owen Holland; Michael Wheeler, eds. *The Mechanical Mind in History*. Cambridge, MA: The MIT Press, 2008.

Ingold, Tim. *Lines: A Brief History*. New York, NY: Routledge, 2007.

Irvine, Andrew. "Russell's Paradox" in *Stanford Encyclopedia of Philosophy*, ed. E. Zalta (Summer 2009), online edition: `http://plato.stanford.edu/archives/sum2009/entries/russell-paradox/`.

———. "*Principia Mathematica*" in *Stanford Encyclopedia of Philosophy* (Winter 2010), online edition: `http://plato.stanford.edu/archives/win2010/entries/principia-mathematica/`.

Jesiek, Brent. "The Origins and Early History of Computer Engineering in the United States" in *IEEE Annals of the History of Computing*, Vol. 35, No. 3 (July-September 2013): 6 - 18.

Jones, Matthew. *The Good Life in the Scientific Revolution: Descartes, Pascal, Leibniz and the Cultivation of Virtue*. Chicago, IL: Chicago University Press, 2006: 229 - 266.

Johns, Adrian. *The Nature of the Book: Print and Knowledge in the Making.* Chicago, IL: University of Chicago Press, 2000.

Kennedy, Hubert. *Peano: The Life and Works of Giuseppe Peano*. Berlin, German: Springer, 1980.

Kirschenbaum, Matthew. "Editing the Interface: Textual Studies and First Generation Electronic Objects" in *Text: An Interdisciplinary Annual of Textual Studies*, Vol. 14 (2001): 15 - 51.

———————. *Mechanisms: New Media and the Forensic Imagination.* Cambridge, MA: The MIT Press, 2012.

Klein, Ursula. *Experiments, Models, Paper Tools: Cultures of Organic Chemistry in the Nineteenth Century.* Stanford, CA: Stanford University Press, 2003.

Kleiner, Israel. "Rigor and Proof in Mathematics: A Historical Perspective" in *Mathematics Magazine*, Vol. 64, No. 5 (1991): 291 - 314.

Kline, Morris. *Mathematical Thought from Ancient To Modern Times*, Vols. 1, 2, 3. Oxford, U.K.: Oxford University Press, 1972.

———————. *Mathematics: The Loss of Certainty.* Oxford, U.K.: Oxford University Press, 1980.

Kline, Ronald. "Cybernetics, Automata Studies, and the Dartmouth Conference on Artificial Intelligence" in *IEEE Annals of the History of Computing*, Vol. 33, No. 4 (2011): 5 - 16.

Knorr-Cetina, Karin. *Epistemic Cultures: How the Sciences Make Knowledge.* Cambridge, MA: Harvard University Press, 1999.

Koss, Adele Mildred. "Programming at Burroughs and Philco in the 1950s" in *IEEE Annals of the History of Computing*, (October-December 2003): 40 - 50.

Kuhn, Thomas. *The Structure of Scientific Revolutions*, 3rd Edition. Chicago, IL: University of Chicago Press, 1996.

Lakatos, Imre. *Proofs and Refutation: The Logic of Mathematical Discovery.* Cambridge, U.K.: Cambridge University Press, 1976.

———————. "A Renaissance of Empiricism in the Recent Philosophy of Mathematics?" in *New Directions in the Philosophy of Mathematics*, ed. Thomas Tymoczko. Princeton, NJ: Princeton University Press, 1998: 29 - 48.

Latour, Bruno. "Visualization and Cognition" in *Knowledge and Society: Studies in the Sociology of Culture Past and Present"*, eds. Henrika Kuklick, Elizabeth Long. Greenwich, CT: Jai Press, 1986.

———————. *Science in Action: How to Follow Scientists and Engineers Through Society.* Cambridge, MA: Harvard University Press, 1988.

———————. *We Have Never Been Modern.* Cambridge, MA: Harvard University Press, 1993.

———————. *Reassembling the Social: An Introduction to Actor-Network Theory.* Oxford, U.K.: Oxford University Press, 2007.

Law, John. *Aircraft Stories: Decentering the Object in Technoscience.* Durham, NC: Duke University Press, 2002.

Lenoir, Timothy. "Was that Last Turn the Right Turn? The Semiotic Turn and A.J. Greimas" in *The Science Studies Reader*, ed. Mario Biagioli. New York, NY: Routledge, 1999: 290 - 301.

Leslie, Stuart. *The Cold War and American Science.* New York, NY: Columbia University Press, 1993.

Lévi-Strauss, Claude. *Structural Anthropology.* New York: Basic Books, 1963.

———. The Savage Mind. Chicago: University of Chicago Press, 1966.

Lewis, C.S. "What the Tortoise Said to Achilles" in *Mind*, Vol. 4, No. 14 (April 1895): 278 - 280.

Light, Jennifer. "When Computers Were Women" in *Technology and Culture*, Vol. 40, No. 3 (1999): 455 - 483.

Linsky, Bernard. *The Evolution of Principia Mathematica: Bertrand Russell's Manuscripts and Notes for the Second Edition.* Cambridge, U.K.: Cambridge University Press, 2011.

———. "The Notation in *Principia Mathematica*" in *Stanford Encyclopedia of Philosophy* (Fall 2011), online edition: `http://plato.stanford.edu/archives/fall2011/entries/pm-notation/`.

Livingston, Eric. "Cultures of Proving" in *Social Studies of Science*, Vol. 29, No. 6 (December 1999): 867 - 888.

Longo, Bernadette. "Metaphors, Robots, and the Transfer of Computers to Civilian Life" in *Comparative Technology Transfer and Society*, Vol. 5, No. 3 (December 2007): 253 - 273.

Loyd, G.E.R. "The Conception and Practice of Proof" in *Demystifying Mentalities.* Cambridge, U.K.: Cambridge University Press, 1990.

Lynch, Michael; Steve Woolgar, eds. *Representations in Scientific Practice.* Cambridge, MA: The MIT Press, 1991.

MacKenzie, Donald. "The Automation of Proof: A Historical and Sociological Exploration" in *IEEE Annals of the History of Computing*, Vol. 17, No. 3 (1995): 7 - 29.

———. *Knowing Machines: Essays on Technical Change.* Cambridge, MA: The MIT Press, 1998.

———. "Slaying the Kraken: The Sociohistory of a Mathematical proof" in *Social Studies of Science*, Vol. 29, No. 1 (1999): 7 - 60.

———. *Mechanizing Proof: Computing Risk and Trust.* Cambridge, MA: The MIT Press, 2004.

———————. "Computing and the Cultures of Proving" in *Philosophical Transactions of the Royal Society A: Mathematical, Physical, and Engineering Sciences*, Vol. 363 (2005): 2335 - 2350.

———————. *An Engine, Not a Camera: How Financial Models Shape Markets*. Cambridge, MA: University of Cambridge Press, 2008.

Mahoney, Michael. "The History of Computing in the History of Technology" in *IEEE Annals of the History of Computing*, Vol. 10, No. 2 (1988): 113 - 125, p. 121.

———————. "Finding a History for Software Engineering" in *IEEE Annals of the History of Computing*, Vol. 26, No. 1 (2004): 8 - 19.

———————. "The Histories of Computing(s)" in *Interdisciplinary Science Reviews*, Vol. 30, No. 2 (2005): 119 - 135.

———————. "What Makes the History of Software Hard" in *IEEE Annals of the History of Computing*, Vol. 30, No. 3 (2008): 8 - 18.

———————. *Histories of Computing*, ed. Thomas Haigh. Cambridge, MA: Harvard University Press, 2011.

Mancosu, Paolo. *From Brouwer to Hilbert: the Debate on the Foundations of Mathematics in the 1920s*. Oxford, U.K.: Oxford University Press, 1998.

Martin, C. Dianne. "ENIAC: Press Conference That Shook the World" in *IEEE Technology and Society Magazine*, (Winter 1995/1996): 3 - 10.

May, Kenneth O. "The Origin of the Four-Colour Conjecture" in *Isis*, Vol. 56 (1965): 346 - 348.

Mazzotti, Massimo. "The Geometers of God: Mathematics and Reaction in the Kingdom of Naples" in *Isis*, Vol. 89, No. 4 (1998): 647 - 701.

McCorduck, Pamela. *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. New York, NY: A K Peters, 2004.

Mean, Margaret. "Cybernetics of Cybernetics" in *Purposive Systems,* eds. Hein von Forester, John White, Larry Peterson, John K Russell. New York, NY: Spartan Books, 1968.

Mehrtens, Herbert. "Mathematical Models" in *Models: The Third Dimension of Science*, ed. Soraya de Chadarevian, Nick Hopwood. Sanford, CA: Stanford University Press, 2004.

Menary, Richard, ed. *The Extended Mind*. Cambridge, MA: The MIT Press, 2010.

Mialet, Hélène. *Hawking Incorporated: Stephen Hawking and the Anthropology of the Knowing Subject.* Chicago, IL: University of Chicago Press, 2012.

Milburn, Colin. *Nanovision: Engineering the Future.* Durham, NC: Duke University Press, 2008.

Mindell, David. "Automation's Finest Hour: Radar and System Integration in World War II" in *Systems, Experts, and Computers: The Systems Approach in Management and Engineering, World War II and After*, ed. Agatha Hughes, Thomas Hughes. Cambridge, MA: The MIT Press, 2000.

Misa, Thomas. "Understanding 'How Computing has Changed the World"' in *IEEE Annals of the History of Computing*, Vol. 29, No. 4 (2007): 52 - 63.

Monk, Ray. *Ludwig Wittgenstein: The Duty of Genius.* London, U.K.: Jonathan Cape, 1990.

———. *Bertrand Russell: The Spirit of Solitude, 1872 - 1921.* New York, NY: Free Press, 1996.

Moore, Gregory. "The Emergence of First-Order Logic" in *History and Philosophy of Modern Mathematics*, Vol. 11 (1988): 95 - 135.

Müller-Wille, Staffan; Isabelle Charmantier. "Lists as Research Technologies" in *Isis*, Vol. 103, No. 4 (2012); 743 - 752.

Nagel, Ernst. "The Formation of Modern Concepts of Formal Logic in the Development of Geometry*"* in *Osiris*, Vol. 7 (1939): 142 - 224.

Netz, Reviel. *The Shaping of Deduction in Greek Mathematics: A Study in Cognitive History.* Cambridge, U.K.: Cambridge University Press, 1999.

———. "Linguistic Formulae as Cognitive Tools" in *Pragmatics and Cognition*, Vol. 7, No. 1 (1999): 147 - 176.

———. *Ludic Proof: Greek Mathematics and the Alexandrian Aesthetic.* Cambridge, U.K.: Cambridge University Press, 2009.

Nofre, David; Mark Priestly; Gerard Alberts. "When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950 - 1960," in *Technology and Culture*, Vol. 55, No. 1 (January 2015): 40 - 75.

O'Leary, Daniel. "The Propositional Logic of *Principia Mathematica* and Some of Its Forerunners" in *Russell: The Journal of Bertrand Russell Studies*,Vol. 8, No. 1 (1988), online edition: `http://digitalcommons.mcmaster.ca/russelljournal/vol8/iss1/10`.

Obermiller, Tim. "Top of His Game" in *University of Chicago Magazine*, 1997, Vol. 89, No. 4 (online edition).

Owens, Larry. "Vannevar Bush and the Differential Analyzer: The Text and Context of an Early Computer" in *Technology and Culture*, Vol. 27, no. 1 (1986): 63 - 95.

Parshall, Karen; David Rowe. *The Emergence of the American Mathematical Research Community, 1876-1900: JJ Sylvester, Felix Klein, and EH Moore*. Providence, RI: American Mathematical Society, 1994.

Parson, Charles; Montgomery Link eds. *Hao Wang: Logician and Philosopher* [*Texts in Philosophy* 16]. London, U.K.: College Publications, 2011: 39 - 48.

Petzhold, Charles. *The Annotated Turing: A Guided Tour Through Alan Turing's Historic Paper on Computability and the Turing Machine*. Indianapolis, IN: Wiley Publishing, Inc., 2008.

Pickering, Andrew, ed. *Science as Practice and Culture*. Chicago, IL: University of Chicago Press, 1992.

————————. *The Mangle of Practice: Time, Agency, and Science*. Chicago, IL: University of Chicago Press, 1995.

————————. *The Cybernetic Brain: Sketches of Another Future*. Chicago, IL: University of Chicago Press, 2011.

Polacheck, Harry. "Before the ENIAC" in *IEEE Annals of the History of Computing*, Vol. 19, No. 2 (1997): 25 - 30.

Priestly, Mark. *A Science of Operations: Machines, Logic and the Invention of Programming*. Berlin, Germany: Springer, 2011.

Pycior, Helena. "Early Criticisms of the Symbolic Approach to Algebra" in *Historia Mathematica*, Vol. 9 (1982): 413 - 440.

Reid, Constance. *Hilbert*. Berlin, German: Springer-Verlag, 1970.

Revisto, Sal. *Mathematics in Society and History: Sociological Inquireis*. Dortrecht, Netherlands: Kluwer, 1992.

Rheinberger, Hans-Jörg. "Experimental Systems: Historiality, Narration, and Deconstruction" in *Science in Context*, Vol. 7, No. 1 (1994): 65 - 81.

————————. *Toward a History of Epistemic Things: Synthesizing Proteins in the Test Tube*. Stanford, CA: Stanford University Press, 1997.

————————. "Scrips and Scribbles" in *MNL*, Vol. 118, No. 3 (April 2003): 622 - 636.

————————. "A Reply to David Bloor: 'Toward a Sociology of Epistemic Things"' in *Perspectives on Science*, Vol. 13, No. 3 (2005): 406 - 410

————————. *An Epistemology of the Concrete: Twentieth-Century Histories of Life*. Durham, NC: Duke University Press, 2010.

Richards, Joan. "Augustus de Morgan, the History of Mathematics, and the Foundations of Algebra" in *Isis*, Vol. 78, No. 1 9March 1987): 6 - 30.

————————. *Mathematical Visions: The Pursuit of Geometry in Victorian England*. New York, NY: Academic Press, 1988.

————————. "Rigor and Clarity: Foundations of Mathematics in France and England, 1800 - 1840" in *Science in Context*, Vol. 4, No. 2 (1991): 297 - 319.

————————. "God, Truth, and Mathematics in Nineteenth-Century England" in *The Invention of Physical Science: Intersections of Mathematics, Theology and Natural Philosophy Since the Seventeenth Century*, eds. Mary Jo Nye, Joan Richards, Roger Stuewer. Alphen, Netherlands: Kluwer, 1992: 51 - 78.

————————. "The History of Mathematics and *L'esprit Humaine: A Critical Reappraisal"* in *Osiris*, Vol. 10 [*Constructing Knowledge in the History of Science*] (1995): 122 - 138.

————————. "Historical Mathematics in the French Eighteenth Century" in *Isis*, Vol. 97 (2006): 700 - 713.

Robertson, Douglas. *Phase Change: The Computer Revolution in Science and Mathematics*. Oxford, U.K.: Oxford University Press, 2003.

Rodin, M.; K. Michaelson; G.M. Britan. "Systems Theory in Anthropology" in *Current Anthropology*, Vol. 19 (1978): 747 - 753.

Rodriguez-Consuegra, F.A. "Mathematical logic and logicism from Peano to Quine, 1890 - 1940' in *Companion Encyclopedia of the History & Philosophy of the Mathematical Sciences*, Vol. 1, ed. Ivor Grattan-Guiness. Baltimore, ML: Johns-Hopkins University Press, 1991: 600 - 616.

Rojas, Raúl; Ulf Hashagen. *The First Computers; History and Architectures*. Cambridge, MA: The MIT Press, 2000.

Rosenberg, Daniel. "Data before the Fact" in *"Raw Data" is an Oxymoron*, ed. L. Gitelman. Cambridge, MA: The MIT Press, 20130.

Rosental, Claude. "Certifying Knowledge: The Sociology of a Logical Theorem in Artificial Intelligence" in *American Sociological Review*, Vol. 68 (2003): 623 - 644.

———————. *Weaving Self-Evidence: A Sociology of Logic.* Princeton, NJ: Princeton University Press, 2008.

Rotman, Brian. "Thinking Dia-Grams: Mathematics and Writing" in *Science Studies Reader*, ed. Mario Biagioli. New York, NY: Routledge, 1999: 430 - 441.

———————. *Mathematics As Sign: Writing, Imagining, Counting.* Stanford, CA: Stanford University Press, 2000.

———————. *Becoming Beside Ourselves: The Alphabet, Ghosts, and Distributed Human Being.* Durham, NC: Duke University Press, 2008.

Rowe, David. "Klein, Hilbert, and Göttingen" in *Osiris*, Vol. 5 (1989): 186 - 213.

———————. "Perspectives on Hilbert" in *Perspectives on Science*, Vol. 5 (1997): 533 - 570.

———————. "Anxiety and Abstraction in Nineteenth-Century Mathematics" in *Science Context*, Vol. 17, No. 1/2 (2004): 23 - 47.

———————. "Making Mathematics in an Oral Culture: Göttingen in the Era of Klein and Hilbert" in *Science in Context*, Vol. 17 (2004): 85 - 129.

Sammet, Jean. "Some Approaches to, and Illustrations of, Programming Language History" in *Annals of the History of Computing*, Vol. 13, No. 1 (1991): 33 - 50.

Schaffer, Simon. "Babbage's Intelligence: Calculating Engines and the Factory System" in *Critical Inquiry*, Vol. 21 (1994): 203 - 227.

Secord, Jim. *Victorian Sensation: The Extraordinary Publication, Reception, and Secret Authorship of Vestiges of the Natural History of Creation.* Chicago, IL: University of Chicago Press, 2003.

Sellen, Abigail; Richard Harper. *The Myth of the Paperless Office.* Cambridge, MA: The MIT Press, 2003.

Shanker, Stuart. *Wittgenstein and the Turning Point in the Philosophy of Mathematics.* New York, NY: State University of New York Press, 1987.

———————. "The Appel-Haken Solution of the Four-Colour Problem" in *Ludwig Wittgenstein: Critical Assessments.* New York, NY: Routledge, 2004: 395 - 412.

Shapin, Steven; Simon Schaffer. *Leviathan and the Air-Pump: Hobbes, Boyle, and the Experimental Life.* Princeton, NJ: Princeton University Press, 1985.

Shieber, Stuart. *The Turing Test: Verbal Behavior as the Hallmark of Intelligence.* New York, NY: Bradford Books, 2004.

Shin, Sun-Joo. *The Logical Status of Diagrams.* Cambridge, U.K.: Cambridge University Press, 2009.

Slezak, Peter. "Scientific Discovery by Computer as Empirical Refutation of the Strong Programme" in *Social Studies of Science*, Vol. 19, No. 4 (1989): 563 - 600.

Smith, Richard. "A Historical Overview of Computer Architecture" in *Annals of the History of Computing*, Vol. 10, No. 4 (1989): 277 - 303.

Steingart, Alma. "A Group Theory of Group Theory" in *Social Studies of Science*, Vol. 42, No. 2 (2012): 185 - 213.

Sterne, Jonathan. *MP3: The Meaning of A Format.* Durham, NC: Duke University Press, 2012.

Strathern, Marilyn. *After Nature: English Kinship in the Late Twentieth Century.* Cambridge, U.K.: Cambridge University Press, 1992.

Suchman, Lucy. *Human-Machine Reconfigurations: Plans and Situated Actions [Learning in Doing: Social, Cognitive, and Computational Perspectives].* Cambridge, U.K.: Cambridge University Press, 2006.

Thomsen, Samuel. "Some Evidence Concerning the Genesis of Claude Shannon's Information Theory" in *Studies in the History and Philosophy of Science*, Vol. 40 (2009): 81 - 91.

Thurston, William. "On Proof and Progress in Mathematics" in *Bulletin of the American Mathematical Society*, Vol. 30 (1994): 161 - 177.

Tymoczko, Thomas. "The Four-Color Problem and its Philosophical Significance" in *New Directions in the Philosophy of Mathematics.* Princeton, NJ: Princeton University Press, 1998: 243 - 266.

van Dalen, Dirk. *Mystic, Geometer, and Intuitionist: The Life of L.E.J. Brouwer.* Oxford, U.K.: Clarendon Press, 1999.

Verran, Helen. *Science and an African Logic.* Chicago, IL: University of Chicago Press, 2001.

Voskuhl, Adelheid. *Androids in the Enlightenment: Mechanics, Artisans, and Cultures of the Self.* Chicago, IL: University of Chicago Press, 2013.

Wang, Zuoyue. "U.S.-Chine Scientific Exchange: A Case Study of State-Sponsored Scientific Internationalism during the Cold War and Beyond" in *Historical Studies in the Physical and Biological Sciences*, Vol. 30, No. 1 [Physicists in the Postwar Political Arena: Comparative Perspectives] (1999): 249 - 277.

———. "Transnational Science During the Cold War: The Case of Chinese/American Scientists" in *Isis*, Vol. 101, No. 2 (June 2010): 267 - 377.

Warwick, Andrew. "The Laboratory of Theory: Or What's Exact about the Exact Sciences?" in *The Values of Precision*, ed. Norton Wise. Princeton, NJ: Princeton University Press, 1995: 311 - 351.

_____. *Masters of Theory: Cambridge and the Rise of Mathematical Physics*. Chicago, IL: University of Chicago Press, 2003.

Wellerstein, Alex. *Knowledge and the Bomb: Nuclear Secrecy in the United States*. PhD Dissertation: Harvard University, submitted October 2012.

Wilder, Raymond. "The Cultural Basis of Mathematics" in *New Directions in the Philosophy of Mathematics*, ed. Thomas Tymoczko. Princeton, NJ: Princeton University Press, 1998: 185 - 200.

Wise, M. Norton, ed. *The Values of Precision*. Princeton, NJ: Princeton University Press, 1997.

Wittgenstein, Ludwig. *Remarks on the Foundation of Mathematics*, ed. G. H. Wright, R. Rhees, and G. E. M. Anscombe; trans. G. E. M. Anscombe. Oxford, UK: Blackwell Publishing, 2001 [1956].

Woolfe, Audra. *Competing with the Soviets: Science, Technology, and the State in Cold War America*. Balitimore, ML: Johns Hopkins University Press, 2012.

Zach, Richard. "Hilbert's Program" in *Stanford Encyclopedia of Philosophy* (July 2003), online edition: `http://plato.stanford.edu/entries/hilbert-program/`.