BJHS Themes (2020), **5**, 205–224
doi:10.1017/bjt.2020.10

RESEARCH ARTICLE

Coded conduct: making MACSYMA users and the automation of mathematics

Stephanie A. Dick*

Department of History and Sociology of Science, University of Pennsylvania, 303 Claudia Cohen Hall, Philadelphia, PA 19104-6304, USA
Email: sadick@sas.upenn.edu

Abstract

This article explores an early computer algebra system called MACSYMA – a repository of automated non-numeric mathematical operations developed at the Massachusetts Institute of Technology from the 1960s to the 1980s, to help mathematicians, physicists, engineers and other mathematical scientists solve problems and prove theorems. I examine the extensive paper-based training materials that were produced alongside the system to create its users. Would-be users were told that the system would free them from the drudgery of much mathematical labour. However, this 'freedom' could only be won by adapting to a highly disciplined mode of problem solving with a relatively inflexible automated assistant. In creating an automated repository of mathematical knowledge, MACSYMA developers sought to erase its social context. However, looking at the training literature, we see that the social operates everywhere, only recoded. This article uses the paper-based training materials to uncover the codes of conduct – both social and technical – that coordinated between the users, developers and machines that constituted the system.

> The problem is that one is mixing two worlds – the world of mathematics and the world of programming.
>
> Joel Moses

Users do not come 'ready-made'. The communities that engage with new technological systems must be created and this process is often far more complex than simply teaching people how to use a new tool. Would-be users must also learn to recognize when a new tool is useful. They must develop new ways of thinking and new practices that incorporate that supposed utility. Even when hacking or breaking or appropriating a technology, users must learn to recognize and accommodate its affordances and limitations. We often work just as hard for our technologies as they have ever worked for us.[1] Would-be users must learn to think in new ways, as well as to do certain things. And when would-be users are not centralized in any particular place, this retooling of head and hand must be done at a

1 See Ruth Schwartz Cowan, *More Work for Mother: The Ironies of Household Technology from the Open Hearth to the Microwave*, New York: Basic Books, 1985, for a foundational articulation of this argument.

distance. Technical manuals have historically served as central platforms for attempting user-community formation at a distance. Of course, they run into well-documented and age-old limitations related to the difficulty of fully formalizing technical skill and the importance of face-to-face interaction, which have been the subject of many historical studies.[2] Although these limitations factor in what follows, this article focuses primarily on a different dimension of manuals. Many complex technological systems embody a fundamental tension – between the needs of the developers who produce and maintain them, and the needs of their would-be users. Choices that are convenient for developers and maintainers often translate into cumbersome or counterintuitive experiences for those less familiar with the functionality and underlying design of their tools.[3] Relatedly, streamlined user experiences often come only at great pains to developers and with the extensive reduction and standardization of user expectations.[4] I argue that the manuals explored here were designed, in part, to mediate this tension. They were designed to make users think more like developers. Specifically, they were designed to make mathematicians and mathematical scientists think more like computer programmers.

This article explores a complex technical system called MACSYMA, for 'Project MAC's SYmbolic MAnipulator'.[5] It was an early computer algebra system, a repository of automated non-numeric mathematical operations, developed at the Massachusetts Institute of Technology from the 1960s to the 1980s (at which point it was privatized), to help mathematicians, physicists, engineers and other mathematical scientists around the United States solve mathematical problems. It was an early example of a so-called 'expert system' – a branch of early artificial-intelligence research that aimed to elicit and encode expert human knowledge in order to make it available in automated form.[6] The would-be users in question were told that the system would free them from the drudgery of much

2 Classic examples include Michael Polanyi, *The Tacit Dimension*, Gloucester, MA: Peter Smith, 1966; David Kaiser, *Drawing Theories Apart: The Dispersion of Feynman Diagrams in Postwar Physics*, Chicago: The University of Chicago Press, 2005; Harry Collins, *Tacit and Explicit Knowledge*, Chicago: The University of Chicago Press, 2010; Steven Shapin and Simon Schaffer, *Leviathan and the Air-Pump: Hobbes, Boyle and the Experimental Life*, Princeton, NJ: Princeton University Press, 1986.

3 The history of technology has long focused on innovation and novelty. Recently, some historians of technology have shifted their attention toward maintenance. See Lee Vinsel and Andrew Russell, *The Innovation Delusion: How Our Obsession with the New Has Disrupted the Work That Matters Most*, New York: Random House, 2020.

4 Early digital computers are a perfect example – they were anything but user-friendly. In the 1940s and 1950s, successful computing was always a feat given the unforgiving obtuseness of the machines, the extremely limited capacities of early computer memory and processing, and the tedium and counterintuitiveness of programming in machine languages. Today's personal computers are often touted as a great triumph in 'user-friendly' technological development. Perhaps ironically, however, the 'user-friendly' character of today's personal computers has been the product of decades-long efforts to depersonalize computing, to standardize and discipline users' needs and expectations as much as possible. See Stephanie Dick and Daniel Volmar, 'DLL hell: software dependencies, failure, and the maintenance of Microsoft Windows', *IEEE Annals of the History of Computing* (2018) 40, pp. 26–49.

5 Project MAC, originally for 'the Project on Mathematics and Computation', was the first institutionalization of the study of computing at the Massachusetts Institute of Technology. It was founded in 1963, with funding from the Defense Advanced Research Projects Agency, under the directorship of Robert Fano, and was an early centre for the study of artificial intelligence, time-sharing, human–machine interaction and theoretical computer science. In the late 1960s it became the Laboratory for Computer Science. See J.A.N. Lee, R.M. Fano, A.L. Scherr, F.J. Corbato and V.A. Vyssotsky, 'Project MAC', *IEEE Annals of the History of Computing* (1992) 14, pp. 9–13.

6 Expert systems were, broadly speaking, attempts to reproduce intelligent behaviour in machinery by encoding the expert knowledge. See Edward A. Feigenbaum, 'The art of artificial intelligence: themes and case studies of knowledge engineering', Stanford Heuristics Programming Project Memo HPP-77-25 (August 1977). MACSYMA is cited often as an early example of an expert system, though it was not called as such by its developers.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

mathematical labour. However, this 'freedom' would only be won by adapting to a highly disciplined mode of problem solving with a relatively inflexible automated assistant. Users would be freed from mundane mathematical labour only insofar as they were able to adapt themselves and their problem-solving needs to the system. And the more adept at using the system people became, the more dependent on automated assistance they risked becoming as well. One of the promises that accompanied MACSYMA was that it could give users access to operations in mathematical domains about which they themselves may know very little. This access would bind users to the system for problem-solving capacities that exceeded their own mathematical understanding. Herein lies a tension at the heart of many 'liberating' technologies – they offer freedom but born out of both dependency and discipline. And this was much easier said than done.

Many difficulties of using the MACSYMA system emerged at the interface between two epistemological domains – mathematics and programming. As Joel Moses, who was in charge of the MACSYMA project from 1971 to 1981 put it, 'The problem is that one is mixing two worlds – the world of mathematics and the world of programming.'[7] Given that both inhere in rules and formalisms, these domains may seem like natural bedfellows. The truth is, however, that actual mathematical practice is not nearly as formalized (or formalizable) as idealizations would suggest. And even what is formalized in mathematics is not necessarily or easily automatable. Mathematicians and mathematical scientists have developed particular forms of representation, standards of demonstration, ways of defining and manipulating their objects of interest, many of which do not easily lend themselves to automation. MACSYMA users had to learn to think and work without these, and instead to translate their objects and problems of interest into forms that accommodated the computer. Sometimes, the translation was impossible. But even when possible, it could be difficult and cumbersome, and the manuals spent a lot of time demonstrating how these translations could be done.

In spite of the difficulties, many people were excited about the problem-solving possibilities MACSYMA presented. During the 1970s, the PDP-10 computer that housed the system became one of the most popular nodes on the ARPANET, the defence-funded predecessor to the Internet.[8] Mathematicians, physicists and engineers at government research agencies, universities, hospitals, oil and gas companies and military research laboratories integrated the system into their problem-solving practice. However, this success was bought at a high cost to the development team at MIT. Building the system was labour-intensive enough on its own, but in order to also retool the heads and hands of mathematical scientists around the country to use it and to think about problem solving in the requisite way, the team at MIT also wrote extensive training materials.

The official, roughly three-hundred-page *MACSYMA Reference Manual*, published by the Computer Science Laboratory, catalogued MACSYMA's functionality, and it was written and rewritten, over and again, in some years every few months.[9] Second was the *MACSYMA Primer*, by Moses, which was a basic introduction to the login and command infrastructure of the system, also frequently revised. These two documents shipped to every interested user for five dollars and one dollar for shipping. There was also *An Introduction to ITS for the MACSYMA User* which was primarily written by Ellen Golden,

7 Joel Moses, 'The variety of variables in mathematical expressions', in *Proceedings of the 1977 MACSYMA Users' Conference*, Washington, DC: NASA, 1977, pp. 123–9.

8 Joel Moses, 'Macsyma: a personal history', invited presentation in Milestones in Computer Algebra, Tobago, May 2008, p. 4.

9 Richard Bogen was the primary author; Jeffrey Golden served as editor, concerned with storage management and debugging; Michael Genesereth wrote the appendix on grammar. Pitts Jarvis formatted the manual and arranged printing, and Ellen Lewis (who later married Jeffrey Golden) was thanked for much helpful assistance.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

the project administrator, who was also centrally involved in the production of the *Manual* itself.[10] Golden's knowledge of the system was extensive, but her expertise was obscured by the manual genre that presents hard-won technical understanding as straightforward, authorless, 'specifications'.[11] Golden's *Introduction* explained the timesharing system that enabled multiple users to work with MACSYMA at the same time, and explained how to access the MIT PDP-10 computer through the ARPANET. Golden (then Ellen Lewis) also wrote 'User aids for MACSYMA', a resource documenting the various resources that one might consult in learning to use the system.[12] MACSYMA could prove so difficult to use that there were even articles and conference papers that aimed simply to explain and catalogue the different kinds of difficulty MACSYMA users may encounter. Foremost in this genre were Moses's 'Algebraic simplification: a guide for the perplexed', and Michael Genesereth's 'The difficulties of using MACSYMA and the function of user aids'.[13] All of these constitute the constellation of paper-based training materials from which this article draws.

The MIT team prophesied their own fate as manual writers in a moment of profound optimism, or hubris, about the impact an automated algebraic system would have on mathematical problem solving. In an early description of MACSYMA, they wrote, 'If such a system can be constructed, its impact on applied mathematics would be substantial. Books would still be used, but only for tutorial exposition.'[14] Needless to say, automated systems have not replaced books in mathematics, but the last part of the prophesy did come true – they spent an awful lot of time writing tutorial exposition. The training materials offer a window into the unfolding negotiations between developers, users and machines – negotiations that in fact defined what those categories meant and what those roles entailed.

Everywhere the manuals and tutorials extol the virtues of experimentation. In a subtle acknowledgement of the impossibility of communicating all technical skill in a written form, users were encouraged to 'play around', to 'see what works', to 'try things', and also to 're-read [the manual] from time to time after the user has worked with MACSYMA so that certain parts which were unclear on prior readings will be better understood in the context of increased familiarity with the system'.[15] Learning to use the system effectively was not the same as learning to understand or to work with the underlying mathematics. Although MACSYMA was explicitly meant to automate existing mathematical techniques, many of these came to look quite unfamiliar to users in their automated form. Even MACSYMA developers acknowledged that 'very often MACSYMA produces large, unwieldy results affording little insight'.[16] Users had to cultivate a

10 Ellen Golden, *An Introduction to ITS for the MACSYMA User*, Mathlab Memo #3 (Laboratory of Computer Science, MIT, version revised 14 April 1981; original date unknown.)

11 The broader mechanisms of erasure and undervaluation of women's knowledge, technical capacity and contribution in the history of computing is documented in, for example, Mar Hicks, *Programmed Inequality: How Britain Discarded Women Technologists and Lost Its Edge in Computing*, Cambridge, MA: MIT Press, 2017; Janet Abbate, *Recoding Gender: Women's Changing Participation in Computing*, Cambridge, MA: MIT Press, 2012; Jennifer Light, 'When computers were women', *Technology and Culture* (July 1999) 40, pp. 455–83.

12 V. Ellen Lewis, 'User aids for MACSYMA', in *Proceedings of the 1977 MACSYMA Users' Conference*, op. cit. (7), pp. 277–90.

13 Michael Genesereth, 'The difficulties of using MACSYMA and the function of user aids', in *Proceedings of the 1977 MACSYMA Users' Conference*, op. cit. (7), pp. 291–308; Joel Moses, 'Algebraic simplification: a guide for the perplexed', *Communications of the ACM* (1971) 14, pp. 527–37.

14 William Martin and Richard Fateman, 'The MACSYMA system', *Proceedings of the Second ACM Symposium on Symbolic and Algebraic Manipulation (SYMSAC)*, Los Angeles, 1971, pp. 59–75, 59.

15 The Mathlab group, Laboratory for Computer Science, *The MACSYMA Reference Manual* (Version Nine, second printing, 1977), p. 1.

16 Genesereth, op. cit. (13), p. 301.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

different kind of knowledge – knowledge of the system, of its behaviour, of its capacities, of its other users and uses.[17]

Within the history of technology, interest in 'users' has often accompanied a desire to get beyond a focus on invention, design and the material configuration of machines and to recover the agency of those who take up, modify and transform them. In 'Users as agents of technological change', Ronald Kline and Trevor Pinch push against the notion that technologies determine how they will be used, insisting that 'the use of an artifact or system has not only resulted in unforeseen consequences, but that users have helped to shape the artifact or system itself'.[18] Similarly, Nelly Oudshoorn and Pinch's edited collection *How Users Matter: The Co-construction of Users and Technology* proposes that 'there is no one correct use for technology' by exploring many unexpected uses that people have found for their machines.[19] While this is certainly true, much literature on users risks framing the history of technology as concatenated acts of 'top-down' and 'bottom-up' social construction, without attending to logics and codes that are shared between developers, users and their machines.[20] Centring 'the user' also risks being as reductive as centring 'inventors' or 'innovators'. I, too, want to unpack how use and users shaped the MACSYMA system, but I also attend to the construction of 'user' as a category in the first place, and the relationships that constituted it. Focusing on and ascribing agency to users is not a way to invert power relations when both developers and users operate within a shared logic, with shared codes of conduct. Historian Joy Rankin has argued that the term 'user', 'now synonymous with "end user" or "consumer,"' fails to capture the creative and communal approach to computing that time sharing afforded and instead fashions the concept of 'computing citizens' to capture the 'technical connections among terminals and computers and telephone wires, but more importantly, the social and sociable interpersonal networks' that constituted timesharing communities.[21] The 'codes of conduct' I explore here are the contractual obligations that obtained between 'computing citizens' in the particular polity of the MACSYMA system, and that were folded into the very notion of a MACSYMA 'user'.

Neither MACSYMA users nor developers were ever alone with the machine. To build or use the system required continuous acts of accommodation, translation, communication and calibration, not just between a user and the tool, but between mathematical problems, machines, programming languages and other people. The system worked when these acts generated shared codes of conduct that were both technical and social and it failed when they did not. Diana Forsythe has argued that artificial-intelligence and expert-systems researchers sought to erase the social, to formulate and then encode knowledge and intelligence without context.[22] No doubt this is what they tried to do. However, close reading

17 For other historical examples of this shift from knowledge of a problem domain to knowledge of an automated system see Stephanie Dick, 'AfterMath: the work of proof in the age of human–machine collaboration', *Isis* (2011) 103, pp. 494–505; Hallam Stevens, 'A feeling for the algorithm: working knowledge and big data in biology', *Osiris* (2017) 32, pp. 151–74.

18 Ronald Kline and Trevor Pinch, 'Users as agents of technological change: the social construction of the automobile in the rural United States', *Technology and Culture* (1996) 37, pp. 763–95, 765.

19 Trevor Pinch and Nelly Oudshoorn (eds.), *How Users Matter: The Co-construction of Users and Technology*, Cambridge, MA: MIT Press, 2005, p. 1.

20 Bryan Pfaffenberger, 'The social meaning of the personal computer: or, why the personal computer revolution was no revolution', *Anthropological Quarterly* (1988) 61, pp. 39–47.

21 Time sharing – in which multiple users dial in to shared, centralized computers – represents a historical alternative to the 'personal computer' in which everyone has their own computer in the home, lab or office. See Joy Rankin, *A People's History of Computing in the United States*, Cambridge, MA: Harvard University Press, 2018, pp. 5-6; and Julien Mailland and Kevin Driscoll, *MINITEL: Welcome to the Internet*, Cambridge, MA: MIT Press, 2017.

22 Diana Forsythe, *Studying Those Who Study Us: An Anthropologist in the World of Artificial Intelligence*, Stanford, CA: Stanford University Press, 2001.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

of the MACSYMA users' literature reveals that the social was everywhere, but it had been recoded around and within the system. People interacted with each other and with the machine according to specific codes of conduct – use was always in relation to those codes and the humans, machines and abstractions they held together. The many iterations of printed training material reveal the process through which these shared codes were negotiated, as well as their many overflows, failures and frictions. In particular, they reveal the modes of thinking and problem solving that aligned with MACSYMA's changing capabilities, codes that were meant to align users' and developers' relationships with the machine and with each other.

This article is about why MACSYMA was hard to use, how the category of 'user' was constructed in this context, what users were meant to be and do as captured in that extensive training documentation. In the section 'Plans and situated thinking' (after Lucy Suchman's *Plans and Situated Actions*), I explore the user-as-'planner' who could break a given mathematical problem down into MACSYMA-solvable sub-problems and then program a corellated path through the system to solve it.[23] According to David Alan Grier, this notion of 'planning' referred to 'the process of preparing a list of instructions that describe a set of operations to be set in motion' by the machine, and that the concept 'developed in the 1920s in England in response to the production demands of World War I'.[24] Industrial logics also operate throughout the early automation of intelligence. Jonnie Penn has shown that they were, in fact, the condition of possibility for the belief that human intelligence and knowledge could be abstracted away from their social and embodied context, highlighting that early artificial-intelligence systems didn't 'think' or 'know' like people but rather like corporations.[25] With MACSYMA, economic notions of programming and planning were introduced to the work of mathematical problem solving. In order to work with MACSYMA, mathematicians had to rethink their objects and problems of interest in machine-oriented, economical terms. In that respect, they had to think (plan) like the system's developers.

The section 'Economizing mathematics' explores another dimension of the MACSYMA code that was at once technical, social and industrial. MACSYMA operated within the strict limitations of the computers that hosted it, whose resources were easy to exhaust. Users were instructed, through the *Manual*, the *Primer*, the 'Maxims for the MACSYMA user', and online, not to hoard computing resources. This meant developing new technical knowledge, for example, about which algebraic operations grow exponentially (and therefore risk exhausting available computing memory) under which circumstances. It also meant being aware of how many 'cycles' of computing (roughly the work done to perform one operation) people were using. Users were meant to think of their own problem-solving activities, or 'jobs', in relation to all other activities and uses. This mutual accommodation and its attendant social and technical practices are a part of the 'computing citizenship' that Rankin identifies in the history of American timesharing more broadly. For MACSYMA, economic and industrial thinking was central to this sociality, because it was central to implementation – programmers and developers looked for efficient and optimal encodings and procedures, and users had to learn to think this way as well.[26]

23 Lucy Suchman, *Plans and Situated Actions: The Problem of Human–Machine Communication*, Cambridge: Cambridge University Press, 1987.

24 David Alan Grier, 'Programming and planning', *IEEE Annals of the History of Computing* (2011) 33, pp. 86–8; Devin Kennedy, 'Virtual capital: computers and the making of modern finance, 1929–1975', PhD dissertation in history of science, Harvard University, 2020.

25 See Jonnie Penn, 'Inventing intelligence: on the history of complex information processing and artificial intelligence in the United States in the mid-twentieth century', PhD dissertation in history and philosophy of science, University of Cambridge, 2020.

26 For more on the epistemological stakes of implementation and memory management see Stephanie Dick, 'Of models and machines', *Isis* (2015) 106, pp. 623–34.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

'The difficulty with simplicity' picks out one specific response to resource scarcity that informed the development of the MACSYMA system. One of the central questions for developers was, how to represent mathematical expressions in the system? It turns out that the most economical or effective representational choices *for automation* diverged considerably from the most economical or effective representational schemes *for people*. Here, I explore the competing notions of simplicity that users and developers had to navigate in order to accommodate the machine and each other as another example of new codes of conduct taking shape.

In part to promote a situational awareness of resource consumption and community, there was no privacy in MACSYMA – anyone could access a list of everyone who was currently dialed in and watch what they were doing. This allowed MIT developers to observe, assist (solicited and unsolicited) and manage the behaviour of users. It also allowed users to watch, learn and borrow from one another. There were also several different commands that allowed users to communicate with one another through the system. In this sense, as well, MACSYMA did not represent an erasure of the social, but rather a recoding of it. Surveillance, observation, mimicry and interaction were formalized in the manuals and normalized in the very category of user that these codes of conduct were designed to create.

The conclusion, 'Finally, there are still people!', explores the means of interpersonal engagement available to MACSYMA users. From the ungenerous ':LUSER' command through which a 'lost user' could ask for help from anyone who would answer to the users' conferences that were inaugurated in 1977, the importance of interpersonal interaction among users highlighted the degree to which the necessary skills could not be transferred by paper alone. The developers' belief in automation, however, would not be undone, as they believed even the human consultant would one day be automated. But first, the next section, 'A laboratory for mathematics', situates the system in its historical context.

A laboratory for mathematics

The MACSYMA system was built and maintained under the auspices of Project MAC, and later the Laboratory of Computer Science at MIT, beginning in the 1960s and into the 1980s, at which time the system was privatized, amid some controversy,[27] and licensed to Symbolics Inc.[28] The MACSYMA development team, called the 'Mathlab group', was constituted primarily of mathematics and electrical-engineering faculty and doctoral students who had an interest in computing and in the automation of mathematical knowledge in particular. MACSYMA differed from most earlier computer systems for mathematical problem solving insofar as it was not merely for numerical calculating. It was not just for solving differential equations, performing Fourier transforms or calculating series; it was for manipulating non-numeric, symbolic mathematical systems. MACSYMA allowed users to represent, simplify, substitute, transform and infer with

27 As is still the case, government-funded computer systems can be privatized as long as government agencies retain their right to use the system free of charge. The Bayh-Dole Act that stipulated these conditions was passed just before the transfer of MACSYMA to Symbolics, but it was still uncertain legal territory at the time.

28 LISP, for 'List Processing Language', was used widely in early artificial-intelligence research. First published in 1960 by John McCarthy, it was based on an earlier set of languages, called 'Information Processing Languages', developed at the RAND Corporation for use in automating logical theorem proving. See Dick, op. cit. (26); Mark Priestley, *A Science of Operations: Machines, Logic, and the Invention of Programming*, New York: Springer-Verlag, 2011, esp. Chapter 8.8; Priestley, 'AI and the origins of the functional programming language style', *Minds and Machines* (2017) 27, pp. 449–72. Symbolics, Inc. was created in 1980 by Russell Noftsker, who served for a time as the administrator of the AI group within Project MAC at MIT. Symbolics originally manufactured computers designed to optimally execute LISP programs, and later acquired MACSYMA.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

algebraic expressions and logical propositions and, in so doing, free up its human users to pursue what they believed were more important and 'properly human' cognitive tasks like 'interpretation, analysis, planning and conjecture'.[29]

In this regard, MACSYMA was a part of a large-scale transformation that unfolded during the 1960s across the United States in which computers went from being treated as larger and more powerful versions of the electric calculating machines and human computers that preceded them, to symbolic machines that could manipulate formal systems whether they referred to numbers or not. The first reprogrammable digital computers in the United States were both functionally intended and conceptually understood as numerical data processors. Numbers were input to the computer, instructions were given for the functional and arithmetic manipulation and storage of numbers, and numeric values were also the output. The developers of MACYSYMA, alongside artificial-intelligence and automated-reasoning researchers, as well as those who sought to establish 'computer science' as an academic discipline, instead subscribed to a vision of computers as symbol-processing machines, capable of manipulating any formal system whatever. This vision gained traction during the 1960s and 1970s, as evidenced by the creation of several conferences and journals focused on symbolic, non-numeric and algebraic computing during that time, many serving as platforms for the circulation of research on and with the MACSYMA system.[30]

The Mathlab group that oversaw its development at MIT was named after an earlier algebraic computing system that informed the project. MATHLAB was first introduced at the Fall Joint Computer Conference in 1965 by Carl Engelman, who held an MS in mathematics from MIT and who spent the majority of his career at the MITRE Corporation. Around the same time, two doctoral dissertations were under way at MIT that also sought to automate elements of non-numeric mathematical practice. William Martin completed his PhD in electrical engineering in 1967 by developing the 'Symbolic Mathematical Laboratory', which offered tools for the automated manipulation of mathematical expressions.[31] Moses, who received his PhD in mathematics in 1967, developed two programs – SIN (Symbolic INtegrator) and SOLDIER (SOLution of Ordinary Differential Equations Routine) – for automated integration.[32] Martin and Moses were both hired as faculty immediately following their receipt of doctoral degrees (this was not uncommon in computing at MIT at the time), and continued their research in the automation of

29 Carl Engelman, 'MATHLAB: a program for on-line machine assistance in symbolic computations', *Proceedings of the Fall Joint Computer Conference* (November 1965), pp. 117–26, 117.

30 Of course digital electronic computers are unavoidably numerical. Information is 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, treated as corresponding to the 1s and 0s of Boolean logic. In both of these senses, the mechanism for computing is numerical. The transition from numeric to non-numeric meant that what was referred to by those numbers in memory or by those bits of information need not be numbers only. Systems of correspondence were worked out and interfaces were devised that enable users to input non-numeric information (even typing on a keyboard is an example of this) and to receive non-numeric information out of the computer (all graphical user interfaces – including the appearance of words on the screen – are examples of this). The crucial insight is that in non-numeric processing, the instructions given to the computer need not correspond to well-defined mathematical functions for numbers. But they can correspond to well-defined processes in other domains, including non-numeric formal systems in mathematics that do not reduce to arithmetic. For more on this development see Priestley, *A Science of Operations*, op. cit. (28), pp. 123–56; Stephanie Dick, 'Computer science', in Georgina M. Montgomery and Mark A. Largent (eds.), *A Companion to the History of American Science*, Hoboken: Wiley-Backwell Publishing, 2015, pp. 55–68.

31 William Martin, 'Symbolic mathematical laboratory', PhD dissertation in electrical engineering, MIT, 1967.

32 Joel Moses, 'Symbolic integration', PhD dissertation in mathematics, MIT, 1967. See also Moses, 'Symbolic integration: the stormy decade', *Communications of the ACM* (1971) 14, pp. 548–60.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

non-numeric mathematics, a field that was gaining popularity throughout the growing computing research community in the US. MACSYMA was first devised in 1969 when Martin decided that MIT should launch a larger-scale effort to combine his systems with Moses's SIN and SOLDIER, and Engelman's MATHLAB, and grow from there. Bob Fano, then director of Project MAC, supported the idea, and in May of 1968 submitted the inaugural application for ARPA and Office of Naval Research funding, proposing that symbolic manipulation systems

> represent a present potential for significant assistance to working mathematicians and to scientists and engineers who use advanced applied mathematics ... The time has now come to integrate as much as possible of these mathematical aids into a single comprehensive system for mathematical assistance capable of helping mathematicians and other scientists.[33]

The application was successful and, by the end of the year, MACSYMA development was under way.

The system grew quickly. By 1971 it consisted of 60,000 words of code (a word being a standardized chunk of computer memory in which instructions were stored) and no less than twenty subroutines that could perform mathematical operations at the user's behest.[34] By the mid-1970s, MACSYMA had become so popular and lucrative that it no longer needed ARPA funding, and instead a User Consortium was developed for support.[35] The community which had learned to use the system, integrated it into their problem-solving practice, and in some cases contributed to its capacities, would now pay dues for the privilege. The original members of the consortium were the Department of Energy; NASA; the US Navy; Schlumberger, an oil and gas exploration company that invested heavily in expert-systems research in the 1970s and 1980s; and several universities (though some were exempted from dues).[36] The consortium revealed the suite of institutions that had access – all of which were connected to the ARPANET, and with it, the PDP-10 computer that housed the MACSYMA system. Access and use were infrastructural and institutional, as well as intellectual, feats.

The US Navy made a survey between 1977 and 1979 of the various ways MACSYMA was being used in their research centres across the country. Everything, from 'analysis of circuits of the Sidewinder missile control systems' at the Naval Weapons Center, to 'finding intensity statistics for a multiple-scatter model of wave propagation through a random medium' at the Naval Research Laboratory, to the 'generation of graphs for presentation and inclusion in reports' at the Naval Ship Research and Development Center, was among the hundreds of reported uses.[37] Elsewhere, users were finding applications of MACSYMA's capabilities for everything from the solution of theoretical-physics problems, to the optimization of databases, to the study of medical devices, to the generation of examples and counterexamples for number-theoretic conjectures. The users of MACSYMA cut across disciplinary boundaries and included mathematicians, doctors,

33 Robert Fano, 'Proposal of supplementary research in computer graphics, computer networks, and a computer-based mathematics laboratory', submitted to the Advanced Research Projects agency and the Office of Naval Research, 15 May 1968, MIT Archives, Collection AC268, Box 21, Folder 34, 6–7.

34 Joel Moses, 'Progress report of the Mathlab group', July 1970, MIT Archives, Collection AC282, Box 25, folder 'Papers and Memos of Joel Moses 1966–74'.

35 MIT Archives, Collection AC268, Boxes 22–4.

36 Moses, op. cit. (8), p. 5.

37 L. Kenton Meals, *Use of MACSYMA in the Naval Laboratory Research Community* (January 1977–1978), Computation, Mathematics, and Logistics Department Departmental Report DTNSRDC/CMLD-79/09 (July 1979), pp. 11–17.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

engineers, physicists and others. They were a community of primarily ARPA-funded mathematical scientists at powerful institutions in the military–industrial–academic complex who may not have had other reasons to interact. By the late 1970s, MACSYMA had, in many ways, outgrown even the formidable infrastructure of MIT and it became, in 1981–3, one of the earliest government-funded computer systems to be privatized, at which point the hard-won MACSYMA community became a market and the automated knowledge at work in the system was commodified. Once it had been extracted, packaged, operationalized and centralized, mathematical knowledge could now conveniently be put up for sale.

Plans and situated thinking

MACSYMA was designed to assist in problem solving by performing operations like algebraic simplification, differentiation, symbolic integration, matrix multiplication and so on, in an automated way. The central insight was that many complex mathematical problems can be broken down into series of these various operations. The MACSYMA user's job was to break a problem down into sub-problems, each of which is known to be solvable by the MACSYMA system. This was called 'planning' and it was picked out as one of the uniquely human abilities for which automated tools were to free the human mind.[38] Planning is an industrial term, as Grier, Penn, and Kennedy have shown, entering the world of computing by way of inter- and post-war industrial culture.

According to Genesereth, who completed a PhD in applied mathematics at Harvard University, and worked on the MACSYMA system during the 1970s, a 'typical' (read, 'ideal') MACSYMA user should proceed analytically: 'in solving his problem the user acts in accordance with a standard, high level planning algorithm'.[39] Depending on the complexity of the problem, sub-problems themselves may need to be broken down into further sub-problems executable by MACSYMA, and so on 'until a level is reached containing only MACSYMA commands'.[40] That is, the user takes the problem she wants to solve and codes an algorithmic path through MACSYMA's capabilities that would hopefully culminate in a solution. Genesereth represented planning algorithms as a 'state and transition augmented network (called SATAN)' (see Figure 1).[41] MACSYMA users disciplined their problem-solving practice in order to follow SATAN through the dark mills of automated mathematics.

The practice of breaking down mathematical problems into constituent sub-problems long precedes the modern digital computer. For example, throughout the nineteenth century, machines were manufactured that could perform only addition. Adding machines could solve any problem that could be reduced to a series of additions.[42] Multiplication is such a problem. In order to multiply two numbers together, you just need to be able to add and count (add 1) each time you've added. MACSYMA could do a lot more than just add but the principle was similar. MACSYMA also allowed people to call on operations that were more difficult by hand or that they may not have the mathematical knowledge to execute on their own.

38 Engelman, op. cit. (29), p. 117.

39 Genesereth, op. cit. (13), p. 297.

40 Genesereth, op. cit. (13), p. 297.

41 Genesereth, op. cit. (13), p. 297.

42 On the history of nineteenth-century calculating machines, and their relation to industrialization, see Lorraine Daston, 'Enlightenment calculations', *Critical Inquiry* (1994) 21, pp. 182–202; Simon Schaffer, 'Babbage's intelligence: calculating engines and the factory system', *Critical Inquiry* (1994) 21, pp. 203–27; Matthew Jones, *Reckoning with Matter: Calculating Machines, Innovation, and Thinking about Thinking from Pascal to Babbage*, Chicago: The University of Chicago Press, 2016.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

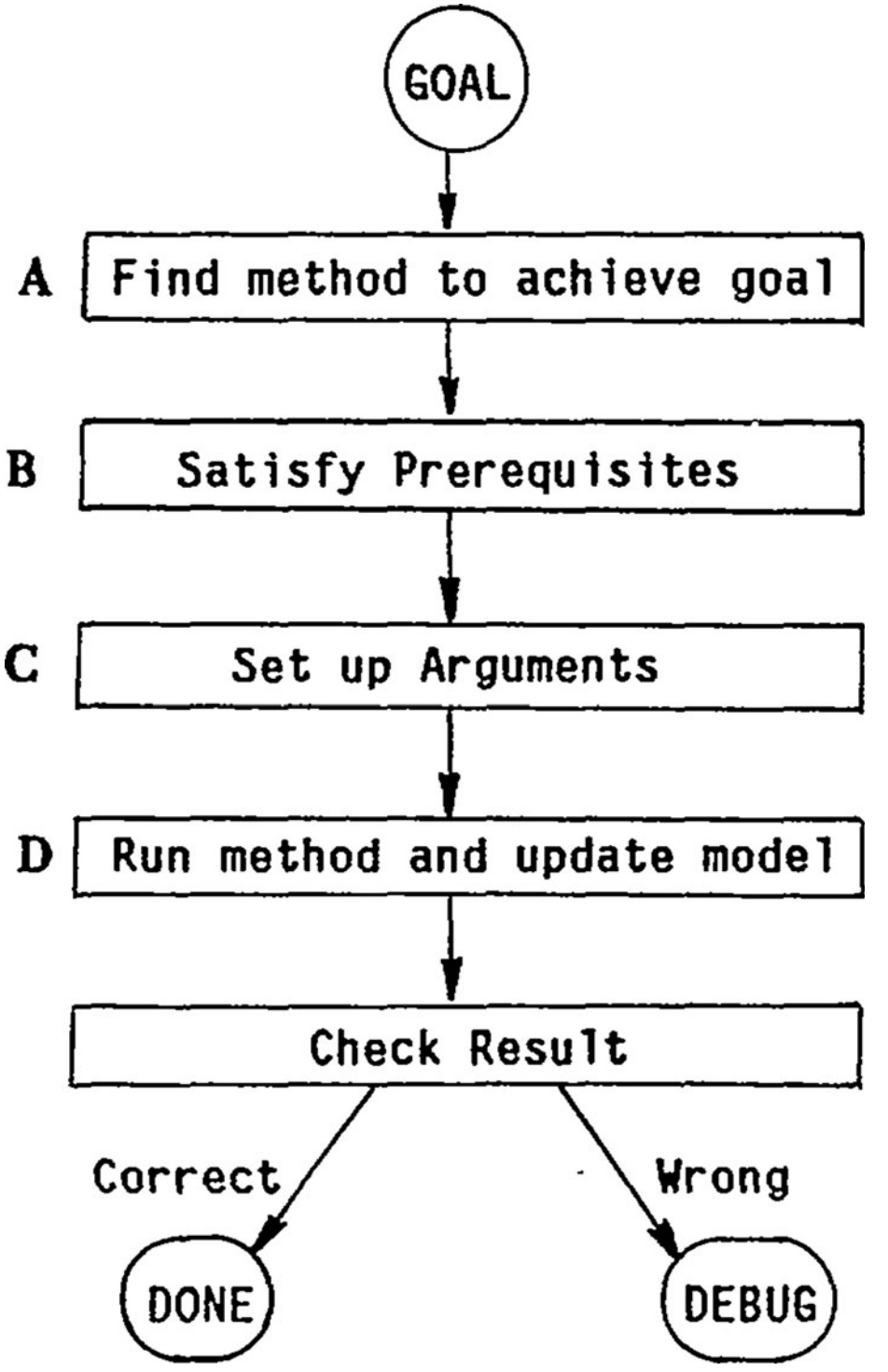

Figure 1. Diagram of the SATAN planning strategy. The original caption reads 'A flowchart for the "typical" user's planning strategy'. Michael Genesereth, 'The difficulties of using MACSYMA and the function of user aids', *Proceedings of the 1977 MACSYMA Users' Conference*, NASA, 1977, pp. 291–308, 297.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

For example, one may wish to minimize a convex function (a bowl-shaped curve) $f(x)$ – that is, find the input to that function that outputs the lowest number on the curve (that input is called the 'minimizer' of the function). This task can be broken down into steps that MACSYMA can execute. Thanks to the fact that $f(x)$ is convex, a point x is known to be a minimizer of f if, and only if, the derivative of f is equal to zero at x. In order to solve this problem, one can first take the derivative of $f(x)$ and then solve for any value x where the derivative $f'(x)$ is equal to 0. Once any point x has been found such that $f'(x) = 0$, this point will be a minimizer of f and the problem is solved. Here, the two sub-problems are (1) taking the derivative and (2) solving the equations. This process can be solved on paper, but it can be tedious. MACSYMA could do derivatives and solve specific equations, and so a solution to this problem can also be produced by translating this plan into the appropriate executable MACSYMA commands, beginning with

```
DIFF(EXP(F(X), X, 1));
```

The function 'DIFF(exp, v1, n1, v2, n2, ...) differentiates [the] exp[ression] with respect to each vi, ni times'. Next, a user could call the SOLVE command in order to get MACSYMA to solve $f(x)$ wherever $f'(x) = 0$.[43] Here, the drudgery of executing these steps was replaced by the drudgery of planning the sub-problem sequence and programming MACSYMA to execute it, in hopes that time is spared in the end. 'In using automatic, recursive rule application, the user is sacrificing the effort necessary to control MACSYMA to eliminate the drudgery of applying the rules himself'.[44] Planning – coming up with a series of MACSYMA operations that would solve a given problem – and programming – translating that series into the requisite commands – were the primary tasks of the MACSYMA user, and its utility corresponded to one's ability to encode problem solving this way.

However, there was no algorithm for coming up with a problem-solving algorithm. One of the 'Maxims for the MACSYMA user' found in the *Primer* was 'The user should anticipate a certain amount of trial-and-error in many calculations in order to determine a good sequence of operations for obtaining the solution.'[45] The *Manual* explained each particular capability of the system and how to call it so users could plan problem-solving paths. The *Manual* is also peppered throughout with suggestions about the effects of *ordering* and *combining* certain operations. For example, some of MACSYMA's 'most powerful and versatile commands' were those for the evaluation and simplification of algebraic expressions. Calling EV(exp, arg1, ..., argn), for example, 'evaluates the expression *exp* in the environment specified by the *argi*'. However, for certain problems, it may make sense to simplify an expression before evaluating it, or to 'resimplify' it without 'reevaluating'. As such, NOEVAL – a function that 'suppresses the evaluation phase of EV' – the *Manual* states, 'is useful in conjunction with the other [argument specifications] in causing *exp* to be resimplified without being reevaluated'.[46] Or, another example, the function XTHRU(*exp*) 'combines all terms' of an expression over a common denominator. 'It can be better', the *Manual* notes, to use XTHRU before calling another function RATSIMP(*exp*) (which 'rationally simplifies' a given expression) 'in order to cause explicit factors of the gcd of the numerator and denominator to be canceled thus simplifying the expression to be RATSIMPed'.[47] That is, XTHRU does one particular part of the simplification that makes the next part more effective and efficient. These operations were

43 Mathlab Group, *MACSYMA Reference Manual*, Version Nine, p. 69.
44 Genesereth, op. cit. (13), p. 303.
45 Mathlab Group, *MACSYMA Primer*, 5 September 1977 version, p. 24.
46 Mathlab Group, op. cit. (45), p. 51, my emphasis.
47 Mathlab Group, op. cit. (45), pp. 57, 61.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

specified *in relation to others*, to signal the importance of sequential thinking – of planning, especially where resource consumption was concerned.

The system included many hundreds of operations and variations like these and, as such, charting possible and optimal paths and combinations could be difficult:

> In a programming system such as MACSYMA there are often many ways to go about solving a given problem as well as many constraints and frustrations which must be dealt with. Some ways will not succeed due to space or time constraints and others may work but may be unnecessarily slow. Frequently *a better understanding of the computer facilities will lead to a reformulation of the problem* lending itself to a much improved solution.[48]

The goal was not simply to learn to use MACSYMA; it was to see mathematical problem solving through the lens of the system's capabilities such that one could be a better planner, recognize dimensions in a problem one might not have been looking for before, and develop a sense of what sequences of MACSYMA operations were effective in different problem domains.

Each MACSYMA operation was meant to be an encoding of existing mathematical practice, the system imagined as a collective repository of the things mathematicians know how to do. MACSYMA thus embodied a theory of knowledge – specifically that it can be made both explicit and procedural for the purposes of automation.[49] Martin and Fateman, the two primary drivers of MACSYMA's development, mused in 1971, 'Our rough hypothesis is that a mathematician knows perhaps 10 000 mathematical facts. For example, if a student learned four facts an hour, four hours a day, five days a week, nine months a year for forty years, he would learn some 12 000 facts.'[50] Most of those facts, they believed, took the form of operations, of 'how-tos': mathematicians know *how to* integrate, they know *how to* simplify algebraic expressions, they know *how to* take the limit of a function. In general, expert-systems practitioners theorized knowledge as procedural in this way, believing it could be translated into commands of the kind that can be given to a computer. But herein lay one of the core difficulties for making use of the system.

Genesereth proposed that MACSYMA was grounded in a 'conception of mathematical knowledge as a body of programming rules'.[51] In trying to explain why many users struggled with the system, he suggested that 'a user may define his operators by the identities they satisfy, but MACSYMA insists on function definitions and unidirectional replacement rules'. That is, mathematicians may define their objects by the properties they have or the constraints they satisfy, but MACSYMA requires instead a *procedure* for creating that object or for recognizing instances where it can be replaced with some equivalent. As a simple example, one definition of an even number might be, 'n is even if, and only if, there exists a number m such that $n = 2m$'. All even numbers will satisfy this property, but this definition doesn't offer an explicit procedure for creating even numbers or establishing whether a given number is even. One such procedure might be 'divide n by 2. If there is no remainder, then n is even'. This is a trivial example, because the transition from a property-based definition to a procedural one is somewhat obvious. However, these translations were often quite difficult, and in some cases impossible. Mathematical knowledge could only be 'implemented in MACSYMA as variable

48 Mathlab Group, op. cit. (45), p. 1, my emphasis.

49 For a history of the related field of automated theorem-proving see Donald MacKenzie, *Mechanizing Proof: Computing, Risk, and Trust*, Cambridge, MA: MIT Press, 2001.

50 Martin and Fateman, op. cit. (14), p. 59.

51 Genesereth, op. cit. (13), p. 302.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

values, function definitions, and TELLSIMP rules, etc., rather than as a set of mathematical definitions and constraints'.[52] MACSYMA development involved the translation of mathematical knowledge and practice into programming rules, and that required users to learn to think in these terms as well. Planning meant not only translating a given problem into MACSYMA-executable sub-problems and commands, but also thinking of the objects and relations at stake in a given problem procedurally in the first place. This was part of how MACSYMA users learned to think like developers, like planners, like programmers. And ideally they would be able to contribute their own knowledge to the system, in the requisite, procedural, operationalized form.

Users were encouraged to encode and operationalize their own knowledge, and attach it to the system in the so-called 'SHARE directory':

> The SHARE directory contains programs, information files, etc. which are of interest to the MACSYMA community. Most files on SHARE; are not part of the MACSYMA system per se and must be loaded individually by the user ... Many files on SHARE; were contributed by MACSYMA users, and all MACSYMA users are encouraged to do so.[53]

Users were not only meant to think like developers, but ideally to become them, able to translate their own knowledge and needs into the system. One function of the ever-updating *Reference Manual* was to catalogue the collective code contributed by those users and make it available to others. Each SHARE directory module was introduced by the command a user would type to call it forth. For example, 'INTSCE LISP contains a routine, written by Richard Bogen, for integrating products of sines, cosines and exponentials of the form EX(A*X+B)*COS(C*X)^N*SIN(C*X)^M. The call is INTSCE(*expr,var*).'[54] Little information was given for each contribution besides the author's name, the call command and a couple of examples. The directory tracked how successfully users' knowledge and needs had been shoehorned into the affordances of the system, packaged for automation. It was a repository of shared codes.

Economizing mathematics

The 'ideal user' of the MACSYMA system adopted as much as possible the perspective of a developer. The previous section focused on the translation of mathematical knowledge and practice into MACSYMA-executable operations. This section turns to the awareness of resources that users were also meant to share with developers, who must understand and accommodate the limitations of the system. The training materials aimed to instill a sense that computing resources were both scarce and shared, and thus everyone had to adopt a certain etiquette of consumption.[55] Throughout the user aids, economic language was used to describe the project of problem solving with MACSYMA – computing operations were 'jobs' and resource-reducing plans 'paid'. Part of thinking like a developer was thinking economically in this way.

Ellen Golden highlighted many of MACSYMA's limitations in *An Introduction to ITS for the MACSYMA User*. For example, the system 'will permit you to have up to eight jobs at one time, but since the system can only accommodate about 120 jobs for all users

52 Genesereth, op. cit. (13), p. 302.

53 Mathlab Group, op. cit. (43), p. 209–30.

54 Mathlab Group, op. cit. (43), 222.

55 The negotiation of shared resources in time-sharing communities is explored extensively in Rankin, op. cit. (25).

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

(including the systems jobs), it is not recommended to have more than two or three'.[56] She continues:

> Occasionally it may be necessary to logout in order to switch terminals, or to permit someone else to use your terminal, but you may have a job you do not want to lose, or which hasn't finished running. The thing to do is to detach yourself, or to disown the job ... Disowning running jobs is inconsiderate, however. If a disowned job wants to print out, it will not be able to and will just stop. This defeats the purpose of its existence and takes up a job slot in the system. What this means is you should only disown running jobs when it is absolutely necessary.[57]

Paying attention to a job's impact on the system as a whole – how many cycles, how much memory it consumed, was a shared code of conduct. Speaking of his experience using MACSYMA at Harvard University in the 1980s, number theorist Fernando Gouvea indicated that 'the hardest part is that we were all using shared computers with not that much capability which meant that you could easily do things that would get you in trouble, that would get the systems administrator screaming at you – "you shouldn't be using all our cycles!"'[58] Whereas most end users of today's computing systems can operate in relative isolation and ignorance of what others are doing, or of the overall systems and networks in which they are consuming resources, MACSYMA users were made aware that they operated in an ecosystem of shared and limited resources, and had to behave accordingly.

In addition to situating one's 'jobs' within the MACSYMA economy, users were also trained to recognize and anticipate excessively consumptive dimensions of their algorithmic planning. For example, two 'Maxims for the MACSYMA user' in the *Primer* referred to the exponential growth of certain computations: 'A common tendency for a beginning user is to needlessly generalize a problem and thus cause inevitable exponential growth ... The user should be aware of the types of calculations which in the general case have exponential growth.'[59] This is to say that it might be possible to easily execute a particular simplification or evaluation of an expression, matrix or Taylor series. However, a user who wanted to evaluate more than one case might 'consider obtaining the determinant of the general case' and this could lead to an explosion of data that would quickly exhaust the available memory. In planning a problem-solving course through the system, some 'intermediate swell' was usually inevitable, but good planning involved cost-cutting decisions all the way through: 'it always *pays* to think before one tries a powerful method', 'it *pays* to reduce the number of variables in a problem as much possible', 'it *pays* if one can reduce the degrees of the variables', 'it sometimes *pays* to convert all expressions to the internal rational function form', and so on.[60] Many of the 'Maxims for MACSYMA users' evidenced the economic and industrial thinking that users should bring to bear on their problem-solving plans. Even though MACSYMA was meant to offer automated versions of known techniques, the system could pursue them to quite inhuman lengths, exceeding what users would have experienced with pencil and paper. As such, they had to develop intuitions for the behaviour of the inhuman execution of familiar techniques and economize accordingly.

56 Golden, op. cit. (10), p. 11.

57 Golden, op. cit. (10), pp. 12–13. 'Disowning' a job would leave it running, but without attachment to input/output from a user's specific terminal.

58 Fernando Gouvea, interview with the author.

59 Mathlab Group, op. cit. (45), p. 24.

60 Mathlab Group, op. cit. (45), p. 24, my emphasis.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

MACSYMA users had to adopt the economic thinking required of developers, both to situate their own goals within the economy of jobs, and to develop the most economical plans they could for solving a given problem. In this way, too, users had to translate their problem-solving needs to accommodate the limitations of the system and one another.

The difficulty with simplicity

One of the most notorious economical concerns in MACSYMA's history related to representation. The question of algebraic simplification epitomizes the tension between users' needs and developers' needs. Simplification is all about representation – there are uncountably many different ways to represent mathematical expressions, and mathematicians choose to work with the symbolic and formal systems that they find most amenable to their problem-solving needs. A primary activity of mathematical scientists is to transform algebraic expressions into amenable forms, to perform substitutions and transformations. For example, 'Many symbolic calculations take the following form: One starts with some equations such as $y = g(h)$, $z = h(x)$, $f = x^2 + y^2 + z^2$, and expressions such as E: $\sum_{i=0}^{5} c_i x^i$. Later one substitutes such questions and expressions into another expression such as $[(\partial f/\partial x)^2 + 2E^2]/f^3$. Then one attempts to simplify the resulting expression.'[61] This was precisely a set of practices that MACSYMA was designed to help with. In manipulating, substituting and simplifying expressions in this way, mathematicians might seek to understand expressions better, to see how they behave, to understand relationships. However, at the heart of this practice are sets of assumptions about what simplicity is, and this turned out to be a controversial subject.

In 'Simplification: a guide for the perplexed', Moses explained why MACSYMA was designed as it was, and how users needed to think differently about simplicity in order to make sense of, and make use of, the system:

> Simplification is the most pervasive process in algebraic manipulation. It is also the most controversial. Much of the controversy is due to the difference between the desires of a user and those of a system designer. The user wants expressions which he can comprehend ... which usually means that the expressions presented to the user should be small. The designer wants expressions that can be manipulated with great ease and efficiency.[62]

These two desires can be in tension. Designers have to consider the affordances of their machines – specific economies of computer memory and processing time – that are quite foreign to those whose practice was grounded in paper and pencil. Users were expecting to find automated versions of their paper-based practices, and instead they found computer-oriented versions that were unfamiliar and unruly.

For example, expression [1] $sin(4\theta)$ and expression [2]

$$4\frac{1-\cos^2(\pi/2-\theta)/\cos^2(\theta)}{1+\cos^2(\pi/2-\theta)/\cos^2(\theta)}\cos(\theta)\cos(\pi/2-\theta)$$

are equivalent. They return the same result no matter what value θ has. We might prefer to work with [1] because it is smaller, easier to read, more intuitive, easier to calculate. However, MACSYMA designers might prefer [2]. In implementing a large-scale system like MACSYMA,

61 Moses, op. cit. (13), p. 529.
62 Moses, op. cit. (13), p. 527.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

it could be more economical to implement only one function, or operation, when possible. For example, it might be more economical to do all trigonometric calculations in terms of cosine, rather than implementing cosine, sine, tangent, cotangent and so on. (This is possible because $sin(x) = cos(x - \pi/2)$, and other trigonometric functions can be represented in terms of one another as well). While [2] is not the simplest representation of [1] in terms of cosine, it could be more amenable to the built-in trigonometric manipulations that MACSYMA can do.

For MACSYMA designers, simplicity was a function of implementation, not visual conciseness or ease of use. But these design features made the system hard to use: 'One of the most common complaints of users of algebraic manipulation systems is that the expressions obtained as results of a calculation are incomprehensible and therefore essentially useless.'[63] MACSYMA designers worked continuously to accommodate users and to produce modules that could translate between the forms of expressions that the computer could most easily work and those that users would most like to see on a screen. But 'simplification for the sake of comprehension' was, for the developer, just one form of simplification among many. MACSYMA's power derived in part from design choices that adhered to other definitions of simplicity, forms that economized for resources other than those used in paper-and-pencil-based practice. Breaking a problem down into MACSYMA-executable sub-problems was, among other things, a literal act of translation in which 'the user must translate his problems into MACSYMA's terms'.[64]

A designer had to decide 'how he will represent expressions, what changes of representation his system will perform automatically, which of these automatic transformations he will let the user override and modify, and what additional facilities for simplifying expressions his system will have'.[65] There were inherent trade-offs in these decisions concerning how many forms of representation a system can handle, and how much freedom users have in choosing their own forms of representation. Not all problems translated easily into MACSYMA's terms, and some could not be translated.

Genesereth included the friction between users' representational choices and MACSYMA's in his taxonomy of 'Difficulties with the MACSYMA system'. He called them 'communication difficulties', the 'result of the difference between the primitive objects, actions, and relations in the user's problem and those provided by the system.'[66] Often a user simply did not know how to reformalize their problem into MACSYMA's terms. Other times, however, there was no 'mapping between the primitives of the user's problem and their representation in MACSYMA'. In the language of expert systems, MACSYMA lacked the 'necessary expertise' to solve such problems. MACSYMA users, therefore, had to develop familiarity not just with the capabilities of the system, but with the representational structures required or permitted by its various operations. Designers and users of the system had to think in terms of new economies, develop different intuitions for problem solving, and learn new representational forms in order for MACSYMA to be a useful system. These, too, were shared codes of conduct that triangulated between users, developers and the machine.

'Finally, there are still people!'

Ironically, in spite of the rejection of tacit knowledge implied by the project of fully automating mathematical knowledge, the MACSYMA community depended on human

63 Moses, op. cit. (13), p. 529.
64 Genesereth, op. cit. (13), p. 300.
65 Moses, op. cit. (13), p. 530.
66 Genesereth, op. cit. (13), p. 300.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

interaction and informal communication for the transfer of technical skill. In addition to the thousands of pages of training materials that circulated, users still struggled to think like programmers, to plan, to break down problems through the lens of MACSYMA capabilities, and to work with the forms of representation the system demanded. There were likely many obstacles to this transition – background, interest, exposure to computing technology, access to community and infrastructure, and more beyond what I can explore here. As such, people supplemented paper in the making of this community in many ways, or, as Ellen Lewis put it at the end of her report on the various aids for MACSYMA users, 'Finally, there are still people!'[67]

Harry Collins and Diana Forsythe each observed that artificial intelligence sought to remove knowledge from its social context.[68] Artificial-intelligence and expert-systems researchers theorized knowledge as something that could be elicited, formalized, automated, and as such something that did not inhere in a body or community or context. However, this position is undone in practice, where 'the social' is not erased by automation at all but rather rearranged around it, mediated, obscured and encoded within it.

First of all, as was common in early time-sharing systems, there was no privacy in MACSYMA. Any online user could request a list of everyone who was online at a given time and could even observe the jobs they had under way. The MIT development team would often observe users and offer assistance if they appeared to be having trouble. Genesereth wrote that these human consultants 'proved to be its most effective user aid'.[69] Lewis too emphasized,

> A very useful aid to a MACSYMA user is the ability to communicate on-line and receive assistance in this fashion. This is the best way to get help … because a MACSYMA programmer at MIT can have access to your current MACSYMA and examine your expressions. He (or she) can also use your console remotely to demonstrate various solutions for you.[70]

While the development team was overwhelmingly male, Lewis herself played a huge role in maintaining the MACSYMA system, handling incoming and outgoing correspondence, writing, typesetting, editing and distributing the documentation, and in this way articulating and documenting the roles of 'user' and 'developer' as they took shape. She was (as indicated here) also sometimes the one offering online help. There are still people, but their roles and genders and labour were so easily hidden in these encoded relationships.

There were also multiple commands that users could call to connect with other users. One command, added after the system was made available outside the MIT community, was :SEND ("message"); with which users could ask for help from any other users who were logged in. Another, less generous, command permitted users to request help mid-programming: 'If you feel you are hopelessly "lost" there is a program called LUSER which will send a message requesting help for you. All you have to do is type :LUSER (followed by a carriage return, of course).'[71] The pejorative command signals the often cruel culture of competence signalling that served (and still serves) to police the borders of technical communities.

MACSYMA-mediated community communication and the mountains of paper-based training materials, however, ultimately did not substitute for the face-to-face

67 Lewis, op. cit. (12), p. 290.

68 Forsythe, op. cit. (22); Harry Collins, *Artificial Experts: Social Knowledge and Intelligent Machines*, Cambridge, MA: MIT Press, 1990.

69 Genesereth, op. cit. (13), p. 299.

70 Lewis, op. cit. (12), p. 19.

71 Golden, op. cit. (10), p. 50.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

development of shared codes of conduct. Some members of the Mathlab group toured user institutions, like the US Navy, to offer in-person courses on MACSYMA use. The MACSYMA User's Conference was inaugurated in 1977 to bring users and developers together. Many of the training documents discussed here, like Lewis's 'User aids for MACSYMA' and Genesereth's 'The difficulties of using MACSYMA and the function of user aids', were first presented at the Users' Conference. The importance of informal human communication and face-to-face communication in developing the MACSYMA community is perhaps ironic given that the belief that knowledge could be extracted from social context, community and embodiment was at the very heart of the project. The social, however, was not erased by their efforts. It was recoded.

MACSYMA codes of conduct were developed to accommodate and coordinate the needs of users, the priorities of developers and the limitations of machines. These codes were both technical and social, ranging from the translation of mathematical problem solving into algorithmic planning to the norms of resource sharing and online communication. Users had to think as much as possible like developers, and all had to accommodate the obtuse limitations of the underlying computing machinery. The reams of paper that constituted MACSYMA user aids enshrined these codes, revealed their changes over time, and sought to carry them across distances, institutions and disciplines. Throughout, the power of this paper-based training regime and its online counterparts was undermined by the comparative efficacy of human help.

Never to be deterred in their ambitions to automate, however, there were computerized versions of the 'help' commands with which users could query the system itself rather than another person. The :HELP( ); command was designed to answer 'How do I <*do something*>' and 'What are the <*arguments, switches*> for <*command*>' questions.[72] MACSYMA would return best-guess answers based on a rudimentary natural-language processor. The :OPTIONS( ); command activated an 'Options Interpreter' that took the name of some MACSYMA command as an input and output a list of things you might do with it.[73] The :DESCRIBE( ); command offered more specific information about other commands, including the inputs and arguments they required. Finally, the :EXAMPLE( ) command and the DEMO directory, in keeping with the print materials' orientation to teaching by example, would output example calls of given commands.

Further, Genesereth lamented,

> Unfortunately, human consultants are a scarce resource and quite expensive. And, as MACSYMA is exported and its user community grows, even more consultants might have to be provided. For this reason, work has begun on the construction of an automated consultant, called the Advisor. This program should be able to converse with the user in English about a difficulty he has encountered and provide advice tailored to his need.[74]

Lewis as well, in describing the automated assistance available in 1977, wrote, 'This HELPer is the beginning of the ADVISOR subsystems which will ultimately take the place of the communication with human advisors for most questions.'[75] Needless to say, the Advisor was never completed in accordance with this vision. Neither did MACSYMA ever become a consolidated body of all mathematical knowledge, formalized as programming rules, though efforts in this direction continue today. The rejection of

72 Lewis, op. cit. (12), p. 281.
73 Lewis, op. cit. (12), p. 282.
74 Genesereth, op. cit. (13), p. 300.
75 Lewis, op. cit. (12), p. 281.

https://doi.org/10.1017/bjt.2020.10 Published online by Cambridge University Press

tacit knowledge that underlay the development of this system undid itself as the paper-training regime failed to fully and easily induct mathematicians into the MACSYMA user community. Instead, far more interesting and complicated social and technical relationships developed, and these constituted the system itself and its users.

Although MACSYMA did not live up to its developers' ambitious visions, it was nonetheless an incredible success. Thousands of people learned to think in new ways, to become planners, to develop economic intuitions – communities spanning navy research facilities, oil and gas companies, universities and hospitals – and more disciplined themselves according to codes shared with each other and with the machine in order to automate their mathematical problem-solving practice. But this recoding was born out of formidable acts of discipline and of reformalization in which the objects and practices of mathematics were conceived in the terms of the system.

So much debate, then and now, orbits the question of what machines can and cannot do, what they will and will not make possible. However, the MACSYMA story highlights the degree to which this question is somewhat beside the point. The real question is, what are people willing to do for their machines? And for the people who build, use and profit from them? What new technical and social codes of conduct take shape as people accommodate the machine and encounter each other through it? The user's manuals and printed aids are our best remaining map of what the negotiation of those new codes looked like in the MACSYMA case and how they took shape at the intersection of mathematics and programming, of people and machines, and of users and developers.

Acknowledgements. I am grateful to my fellow authors for many transformative discussions; to my very helpful reviewers for incisive feedback; to the editors of this volume for guidance and patience; and to Marc Aidinoff, Travis Dick, Mynra Perez Sheldon, Henry Cowles, Oriana Walker, Jonnie Penn, Michael Barany, Joanna Radin, Luke Stark, Lukas Rieppel, Stefan Helmreich, Peter Galison and Etienne Benson for their invaluable perspectives on various drafts. I am also grateful to the reading group at the Max Planck Institute for History of Science, Department II, and to Lorraine Daston and Norton Wise in particular, for their engagement with a version of this material in May 2018.

Cite this article: Dick SA (2020). Coded conduct: making MACSYMA users and the automation of mathematics. *BJHS Themes* **5**, 205–224. https://doi.org/10.1017/bjt.2020.10