Amodern 9: Techniques and Technologies
April 2020

Histories of Techne

ON COMPUTER SCIENCE AS "PROCEDURAL EPISTEMOLOGY"

Mingyi Yu

An influential textbook from the field of computer science, *Structure and Interpretation of Computer Programs* (1985), begins with the provocation that the subject it introduced "is not a science and that its significance has little to do with computers." [1] Rather, its authors, Harold Abelson and Gerald Jay Sussman, sought to establish a more philosophical basis for interest:

> The computer revolution is a revolution in the way we think and in the way we express what we think. The essence of this change is the emergence of what might best be called *procedural epistemology* – the study of the structure of knowledge from an imperative point of view, as opposed to the more declarative point of view taken by classical mathematical subjects. Mathematics provides a framework for dealing precisely with notions of "what is." Computation provides a framework for dealing precisely with notions of "how to." [2]

The textbook outlines the trajectory of an entry-level course Abelson and Sussman developed in the late 1970s at the Massachusetts Institute of Technology (MIT), where it was a requirement for all students majoring in electrical engineering or computer science. Its approach is said to have "revolutionized the landscape of the introductory computing curriculum," shifting emphasis away from the "tyranny of syntax." [3] Indeed, the authors' choice of a less-than-popular programming language, a dialect of LISP called Scheme, as the means of instruction was motivated by its relative syntactic simplicity, with the (intended) effect that "after a short time we forget about the syntactic details of the language (because there are none) and get on with the real issues – figuring out what we want to compute, how we will decompose the problems into manageable parts, and how we will work on the parts" (*SICP*, xvi and 2-4). [4]

In what conditions is it possible to think of "the real issues" involved in computer programming as distinct from the details of computers, machinery which the authors seemingly want "little to do with" – that is, to refer to it as an activity suspended from the day-to-day task of producing efficient (and, of course, commercially valuable) programs? Abelson and Sussman articulate two concerns behind their peculiar stance. First, they insist on "the idea that a computer language is not just a way of getting a computer to perform operations but rather that it is a novel formal medium for expressing ideas about methodology. Thus, programs must be written for people to read, and only incidentally for machines to execute" (*SICP*, xv). In formulating computer science as a sort of media theoretical endeavor, importance is weighted towards the development of interpretative skills. Mastery entails the introjection of a filtering operation, resulting in the cultivation of a faculty of judgment. Participants who successfully complete training are distinguished as much by their ability to take in arcane combinations of symbols, as by their facility at discerning what may be safely ignored and thus excluded from attention: "They

Skip to main content

should be capable of reading a 50-page-long program, if it is written in an exemplary style. They should know what not to read, and what they need not understand at any moment" (*SICP*, xvi).

Their second point more precisely addresses the appropriate terrain for novices who "have had little or no prior formal training in computation." Priority is displaced not only from the acquisition of specific empirical knowledge ("not the syntax of particular programming-language constructs, nor clever algorithms for computing particular functions efficiently"), but also from forms of abstract reasoning ("nor even the mathematical analysis of algorithms and the foundations of computing"). Instead, Abelson and Sussman consider the "essential material" for beginners to be "the techniques used to control the intellectual complexity of large software systems" (*SICP*, xv). Again, execution on machines, whether actual circuitry or the idealized models used in algorithmic analysis, is only of ancillary relevance. While it was common throughout the 1980s to frame the problems posed by computation in terms of complexity, the focus here expressly on "*intellectual* complexity" subtly shifts the implications of the term; not only are programs primarily situated in the mind, as "techniques" they must be practiced, incorporated, and performed. [5] Philip Wadler's 1987 article offering an alternative curriculum criticizes details in Abelson and Sussman's approach but nevertheless concurs with the general outlook adopted in their textbook, stating even more explicitly: "The purpose of the first course is to teach basic principles and develop good habits of thought." [6]

In critical theorization, conceptions of "code" tend to emphasize (or, at least, begin from) an operative or functional dimension, noting that it "can be rendered into machine-readable commands that are then executed," "causes things to happen," and "is the only language that is executable ... machinic first and linguistic second." [7] Self-presentation and identification by programmers, by contrast, dwells on the exact opposite. Further instances of such an attitude are easily found; in this, Abelson and Sussman are not an aberration but the norm. Donald Knuth championed the practice of "literate programming," urging that "instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do." [8] Edsger Dijkstra's characterization of algorithms as a means of "executional abstraction ... to map different computations upon each other" emphasized that "it refers to the way in which we can get a specific computation within our intellectual grip by considering it as a member of a large class of computations" – that is, emphasized the mental process of "abstraction" over the actual "executional" function. [9]

Can programming really be taken seriously as a medium of literacy? What should one make of the inclination of programmers to sound like Heideggerian philosophers meditating on the ready-to-hand, treating their work in terms of an "intellectual *grip*"? A study of media following such prompts would examine neither hardware nor even software proper, but computation's pedagogical apparatus – in other words, such concerns direct media theory towards computation's role in informing a system of inculcation and sense-making, or its operations as a cultural technique. Friedrich Kittler, who first appropriated the notion of cultural techniques in a media theoretical context, also offered reflections on the concept of "code" more appropriate to this direction:

Skip to main content

> The so-called “hidden layers” in today’s neuronal networks present a good, if still trifling, example of how far computing procedures can stray from their design engineers, even if everything works out well in the end. Thus, either we write code that in the manner of natural constants reveals the determinations of the matter itself, but at the same time pay the price of millions of lines of code and billions of dollars for digital hardware; or else we leave the task up to machines that derive code from their own environment, although we then cannot read – that is to say: articulate – this code. Ultimately, the dilemma between code and language seems insoluble. [10]

Kittler’s earlier study of German poetry circa 1800, first published in the same year as Abelson and Sussman’s textbook, had turned away from interpretation of what poems mean towards an examination of the conditions through which meaning-making was instilled as a defining characteristic of subjectivity, examining the “elementary acculturation techniques” of learning to read; the initial translation of the compound word, *Kulturtechniken*, emphasizing the dynamics of “acculturation,” helpfully highlights the processual aspects implied in the production of educated or cultured subjects. [11] His later commentary on code takes up similar concerns, situating it within histories of encoding practices such as encryption. As described and performed by Kittler (who maintained his own sort of “literate” programming practice), computing is a factor directly implicated in cultures of literacy, a matter of being able to “read” and “articulate” what is coded. Indeed, his assessment of instructional primers from the 1800s may well be applied to textbooks on computer programming from the 1970s and 80s: “The project was to replace rote learning with ‘understanding.’” [12]

Addressing programming as a cultural technique recalls this early pedagogical inflection whilst also resonating with subsequent attention to threshold operations that articulate the emergence of symbolic distinction. [13] Such media theory is less concerned with the attributes of mediums *per se*, than with their bearing on conditions of legibility that establish the standards differentiating sense from nonsense, demarcating the literate from the illiterate. Thus, following such approaches, I am less interested here in determining an ontology of code (the defining and definitive attributes of a medium), than in historicizing conceptions of code and placing it within currents of knowledge practices (in what capacities, to what ends, and with what effects does a medium get treated as such). In this sense, my study might further be considered an archaeology of computer programming, akin to Michel Foucault’s attempts “to uncover the regularity of a discursive practice.” [14] An assessment that Abelson and Sussman “revolutionized” the instruction of computing – that the publication of their textbook is most productively interpreted in terms of historical discontinuity – is most compelling when one does not seek to identify a breakthrough in knowledge on hitherto intractable problems. Instead, the rupture that it occasions is the implementation of a normative revision in the regulative ideals to which practitioners aspire, i.e. what “they *should* be capable of” and “they *should* know.” Expertise, as Abelson and Sussman envision it, *should* not just account for the accumulation and possession of information but also consist of dimensions of competency; the pedagogy of computer science is especially interesting as a live experiment on the challenges this entails.

While it is unlikely that the notion of a "computer science" would appear paradoxical to anyone today, perhaps it should. This formulation expresses a project slightly distinct from a straightforward "science of computers." The two terms from which it is comprised are held together in an awkward tension in which each seems to preclude the other. By the 1980s, when Abelson and Sussman amended this name in terms of a doubled negation (*not*-"computer" and *not*-"science"), the discipline which it labelled had experienced more than two decades of anxiety; one could say that any meticulous interest in computing machinery had been considered a threat to scientific propriety ("computer" = *not*-"science"), and conversely that any aspirations to being scientific had entailed a disavowal of the technological substrate of computation ("science" = *not*-"computer").

In its early years, the credibility of this newly formed "computing science" provoked a mixture of suspicion and hostility: a common refrain held that "any science that needs the word 'science' in its name is by definition not a science." [15] John Backus's recollections of early efforts in programming depicts a culture disposed towards circumstantial fixes, appreciative as much of the ingenuity required to conjure up creative solutions (or "hacks") as of its resistance to standards of normativity. His description of the practice as "a black art, a private arcane matter" pointed to the lack of general principles and established conventions that allowed it to escape managerial discipline. "Programming in America in the 1950s," Backus wrote, "had a vital frontier enthusiasm." [16] Plagued by issues of reliability in hardware and written under pressures of limited computational resources like speed and memory, early software was characterized by the prevalence of tricks which worked only on individual machines, resulting in an artisanal (and decidedly unsystematic) mode of production. Historian of computing Nathan Ensmenger has elaborated that in order to achieve "acceptable levels of usability and performance," programmers of that era "had to cultivate a series of idiosyncratic and highly individual craft techniques designed to overcome the limitations of primitive hardware." [17] That is, a prerequisite of effective production was the acquisition of deep practical familiarity with the quirks of devices ("computer"), making the prospect of an autonomous discipline of secure knowledge appear dubious (*not*-"science").

On the other hand, those more professionally-inclined sought to establish a respectable basis for their work. Backus identified the impediment to be an absence of legibility: "Existing programs for similar problems were unreadable and hence could not be adapted to new uses.... Thus each problem required a unique beginning at square one, and the success of a program depended primarily on the programmer's private techniques and invention." [18] Along similar lines, Dijkstra would question in a 1975 speech if the programmer was a "Craftsman or Scientist?"; here, as well, the challenge was treated as a problem of transmission: "To make implicit knowledge explicit and to discuss how to describe skills, so that they can be transferred, implies, if not the birth at least the conception of a new science." [19] It should thus perhaps be no surprise to find Abelson and Sussman claim that "a computer language ... is a novel formal medium." From this perspective, the peculiarities of actual machinery presented the threat of noise corrupting the cleanliness of communication. Dijkstra would later polemically summarize the ethos he defended, comparing physical materialization of machinery to pathological infection: "I don't need to waste my time with a computer just because I am a computer scientist. (Medical researchers are not required to suffer from the diseases they investigate.)" [20] In order to acquire legitimacy as a body of knowledge (“science”), output had to be formalized in machine-independent terms (*not*-“computer”).

My contribution here builds on work by historians of computing detailing the paths through which these modes of expertise were individuated and stabilized, but focuses instead on hesitations and specters – specifically, the persistence of craftsmanship as both an unattainable ideal and looming danger against epistemic legibility amidst the broader field in which knowledge practices were reconceptualized over the postwar era.[21] Dijkstra described “a disastrous blending, viz. that of the technology of the craftsman with the pretence of the scientist”:

> The craftsman has no conscious, formal grip on his subject matter, he just “knows” how to use his tools. If this is combined with the scientist’s approach of making one’s knowledge explicit, he will describe what he knows explicitly, i.e. his tools, instead of describing how to use them! If he is a painter he will tell his pupils all he knows about all brushmakers and all he knows about the fluctuating price of canvas. If he is a professor of computing science, he will tell his students all he knows about existing programming languages, existing machines, existing operating systems, existing application packages and as many tricks as he has discovered how to program around their idiosyncrasies. And in a short while, he will not only tell what the manual says should be punched in column 17 of the first card in order to indicate your choice of priority queue, but he will also tell and explain the illegal punching in column 17 that will place your program in the highest priority queue while only charging you for the lowest priority one.[22]

This doubled affirmation (“computer” *and* “science”) combined the worst attributes of craftsman and scientist, resulting in the systematic communication of trivial details. A rejection of craft did not imply the straightforward affirmation or adoption of its apparent other, science. Dijkstra’s rant suggests the impetus behind Abelson and Sussman’s proclamation of computer science as a “procedural epistemology,” determined to avoid the absurdity of rigorous instruction in contingent specifications – hence, *not*-“computer,” but also *not*-“science.”

The goal of this “revolutionized” pedagogy was instead for students to “have a good feel for the elements of style and the aesthetics of programming” (*SICP*, xv). In this, Abelson and Sussman reiterate the sentiments of their colleague Marvin Minsky, whom they cite in an epigraph:

> A computer is like a violin. You can imagine a novice trying first a phonograph and then a violin. The latter, he says, sounds terrible. That is the argument we have heard from our humanists and most of our computer scientists. Computer programs are good, they say, for particular purposes, but they aren’t flexible. Neither is a violin, or a typewriter, until you learn how to use it.[23]

Minsky’s article explaining “Why Programming Is a Good Medium for Expressing Poorly-Understood and Sloppily Formulated Ideas” (1967) had disputed the notion that computing’s limitations constrained its relevance to domains of “perfect precision” or “rigid rules.” Initially presented at a conference on the topic of “Computers in Design and Communication,” it severely departed from the discussions of possible applications of computing machinery taken up by the other papers there. Whereas, in the examples Minsky provides, the *technology* of a phonograph minimizes the impact of human factors in the operating process, comparing computers to violins implies the opposite, framing it as necessitating training in *techniques*: "To take advantage of the unsurpassed flexibility of this medium requires tremendous skill – technical, intellectual, and esthetic." [24]

Sussman's doctoral dissertation (of which Minsky was a reader) further pursued this line of inquiry. Completed at MIT's Artificial Intelligence Laboratory, the subject matter addressed in his proposal for "A Computational Model of Skill Acquisition" (1973) would be somewhat surprising for work done in the Department of Mathematics were one unfamiliar with the vicissitudes around competency in computing:

> An important property of skill is effectiveness. It wouldn't be enough to memorize all of the facts in the plumber's handbook, even if that could be done. The knowledge would not then be in an effective, usable form. One becomes skilled at plumbing by practice. Without practice it doesn't all hang together. When faced with a problem, a novice attacks it slowly and awkwardly, painfully having to reason out each step, and often making expensive and time-consuming mistakes.
>
> Thus the skill, plumbing, is more than just the information in the plumber's handbook; it is the unwritten knowledge derived from practice which ties the written knowledge together, making it usable.
>
> –Just what is this unwritten knowledge? How is a skilled person different from a knowledgeable, unskilled one?
>
> –What is the process by which a person develops skill through practice?
>
> The research reported in this document is an attempt to provide these questions with precise, mechanistic answers by the construction of a computational performance model, a computer program which exhibits behavior identifiable with skill acquisition. [25]

Still, it remains bizarre that plumbing should be a paradigmatic point of reference for advanced work in mathematics. Sussman elaborates on the topic in a manner strongly resonant not just with the warnings Dijkstra would raise, but also with more recent scholarship from other fields. The stakes set out in his discussion are echoed, for example, in anthropologist Tim Ingold's writings on *The Perception of the Environment* (2000), which highlight the qualities of "fluency" and "dexterity" when maintaining that "intentionality and functionality are immanent in the practice itself, rather than being prior properties, respectively, of an agent and an instrument," with the consequence that "it is not through the transmission of formulae that skills are passed from generation to generation, but through practical, 'hands-on' experience." [26] Likewise, reflecting on *The Craftsman* (2008), sociologist Richard Sennett contends that "skill is a trained practice" encompassing "a dialogue between concrete practices and thinking": "Whoever has tried to assemble a do-it-yourself bookcase following written instructions knows the problem." [27]

Most strikingly (and, indeed, rather puzzlingly), Sussman's research does not simply contrast written knowledge with skillful practice, but also seeks "precise, mechanistic answers" to what would seem to be, by definition, imprecise and extra-mechanistic. In this regard, he makes somewhat more radical propositions than Ingold and Sennett, whose arguments tend to slide into a primitivist celebration of artisanal values as pre-technological or anterior to the alienating effects of modernity. Programming offered itself along the disposition staked by Minsky, as "a Good Medium for Expressing Poorly-Understood and Sloppily Formulated Ideas," collapsing the distinction between mechanism and skill.

These claims were certainly not without controversy, but I am not particularly interested in the veracity or efficacy of the approach. Instead, the adoption of such a position is indicative of the changing epistemological significance of procedures, a mark of shifting discursive conditions. Hence, amidst the emergence of a mathematics based on plumbing, it becomes possible, as well, to state *objections* of the sort that Lucy Suchman would forward:

> Skilled activities like driving proceed in a way that is only derivatively and summarily characterizable in terms of procedures or rules – and such rules as do get formulated are only used when the activity needs for some reason to be explicated, as for instruction, or at times of breakdown when otherwise transparent ways of proceeding need to be inspected or revised. [28]

Suchman's distinction between *Plans and Situated Actions* (1985) drew on her observations at Xerox's Palo Alto Research Center (PARC), where she was embedded as a doctoral student in anthropology. The "planning model," which Suchman considered dominant in research on artificial intelligence and cognitive science, conceptualized purposeful action as the implementation of an underlying design independent of the exigencies of a situation. By contrast, she championed an alternative which accounted for the particular circumstances or "situated"-ness of actions. The influence of Martin Heidegger (via Hubert Dreyfus) is explicit. Understood this way, a plan does not offer a definitive dictation of process but is wielded as an intervention:

> The "unready-to-hand," in Heidegger's phrase, comprises occasions wherein equipment that is involved in some practical activity becomes unwieldy, temporarily broken, or unavailable. At such times, inspection and practical problem solving occurs, aimed at repairing or eliminating the disturbance in order to "get going again." In such times of disturbance, our use of equipment becomes "explicitly manifest as a goal-oriented activity," and we may then try to formulate procedures or rules....
>
> Situated action, in other words, is not made explicit by rules and procedures. Rather, when situated action becomes in some way problematic, rules and procedures are explicated for purposes of deliberation and the action, which is otherwise neither rule-based nor procedural, is then made accountable to them. [29]

My concern in citing these contrasting arguments is not in resolving their claims in favor of one side or the other, but in pointing to shifts in the standing of procedural

Skip to main content

knowledge or “know-how” in its emergence as a positivity – the historical passage through which it becomes liable to truth and falsehood. In the relatively condensed moment of time around the publication of these texts in the 1980s, the question of “procedure” becomes a site which can be interrogated, from which truth can be extracted. To be clear, my argument is that prior to this, a contention like Suchman’s refutation is neither true nor false, but impossible; to reject (or affirm) the adequacy of procedures as skill would have been a type error or category mistake, the result of a misapplication of concepts.

The expression, “procedural epistemology,” is laden with irony, as the procedural is precisely that which has traditionally been excluded from the premises of epistemology. “At the beginning of its history,” Bernard Stiegler has argued, “philosophy separates *tekhnē* from *epistēmē*, a distinction that had not yet been made in Homeric times” [30] ; problems of “how to” were, by definition, considered factors beyond the epistemic realm. It is, of course, not that practices had never existed or been codified, but one could not have taken the gap between rule and action as an object of investigation, as will be found in treatments from the 1970s and 1980s – it was, quite specifically, illegible: “unwritten knowledge,” as Sussman puts it. To cite two examples which have been particularly influential, when Gilbert Ryle introduced the distinction between “knowing how” and “knowing that” in the late-1940s, the former is categorically and axiomatically distinguished from the latter. [31] Similarly, Michael Polanyi’s conceptualization of “tacit knowing” in the 1950s is simply proposed in the negative, as what may not be prescribed explicitly. [32] While these seminal accounts undoubtedly provide the conditioning background and even terminology of subsequent debate, they take as given what others a few decades later painstakingly argue over. For Ryle and Polanyi, the “planning model” is not only false, but nonsensical: there is no possible grounds for them to even begin an argument on this.

Ryle’s argument is especially useful as a point of comparison, as its substantive content largely resembles the distinction subsequently made by Suchman. He writes: “the general assertion that all intelligent performance requires to be prefaced by the consideration of appropriate propositions rings unplausibly.... Overt intelligent performances are not clues to the workings of minds; they are those workings.” [33] Yet, this apparent similarity between their positions derives from a subtle but sharp disparity in their bases of inquiry. As Ryle understood it, his task as a philosopher was merely to clarify category mistakes expressed in the habitual correlating of concepts (sense vs. nonsense), not to stage examinations of empirical correctness (true vs. false). The latter is precisely what Suchman believed her ethnographic account enabled her to do: “Every human tool relies upon, and reifies in material form, some underlying conception of the activity that it is designed to support. As a consequence, one way to view the artifact is as a test on the limits of the underlying conception.” [34]

Even Thomas Gladwin’s short article, “Culture and Logical Process” (1964), cited repeatedly by Suchman as inspiration, speaks from another position. In this text, Gladwin distinguishes “a contrast between two cognitive strategies” towards navigation: (1) the European, which “begins with a single unifying plan.... Almost all the thinking is done in advance of its implementation”; and, (2) the Trukese, in which “each move is successively determined on an *ad hoc* basis. Thinking is continuous, and in our culture we should consider it a series of improvisations." [35] Suchman declares her firm commitment to the latter, arguing that "the essential nature of situated action, however planned or unplanned, is Trukese." [36] Despite also maintaining a clear appreciation for the Trukese strategy, Gladwin's account does not attempt to disprove the former's veracity as a model of cognition, merely its absolute monopoly; he does not subordinate either strategy to the other. In fact, his book, *East Is a Big Bird* (1970), which expands on the earlier article through a more thorough study of neighboring Puluwat navigators, concludes quite overtly: "However diverse the intellectual traditions of the navigator, the sea is a demanding master. No style of thinking will survive which cannot produce a usable product when survival is at stake." [37] For Gladwin's purposes, these models belong to incommensurable cultures, presenting different styles of logic, and thus cannot properly be studied in terms of correctness; it is not the cultural anthropologist who is the arbiter of such truth, but the sea.

The point, again, is not that Suchman has misread her source, but that the discursive environment in which statements regarding "know-how" are possible has shifted. Historicizing the development of computer science within this broader field of knowledge practices demonstrates the contingency of epistemic cultures, thus highlighting how its medial operations stage a revaluation of sense. Programming, as "a novel formal medium" or cultural technique, is not just an efficient portal for procedures, but constitutive of the terrain encompassed by such a concept.

Standard critical narratives tend to think of the historical relation between techniques and technologies as a movement from the former to the latter, i.e. as the attempt "to replace subject-centred skills with objective principles of mechanical functioning." [38] However, the efflorescence of computing as a "medium" (to follow the terminology of computer scientists) problematizes this presumed linear trajectory. Instead of outsourcing expertise, the introduction of machinery counterintuitively led to an increasing preoccupation with dimensions of competency. Despite its roots as a "government machine" developed in support of bureaucratic goals, the computational apparatus was characterized by recalcitrance to organization; one might even find it useful to interpret the emergence of programming as a "black art" as the autophagic collapse of managerial fantasies, wherein the quest for total administration gave rise to the exact opposite, precipitating a surge of pressure onto seemingly ineffable factors beyond the realm of purely explicit rules. [39] Computer science, as a discipline of "procedural epistemology," coalesces under these conditions, not as the merely technical implementation of solutions but amidst shifts in the aspects of knowledge that Dijkstra called "executional abstraction."

Scholars examining the epistemic significance of computing have thus far articulated important insights on its role as a *technology* of simulation, both in established disciplines like physics and new fields like climate science. [40] It seems to me equally crucial to consider computation as the media *a priori* that makes *technique* possible as a form of knowing. That is, far from leading to an eradication of skill, only under conditions of computation does skilled activity becomes epistemologically legible.

Perhaps the clearest demonstration of this may be found in the inception of Polanyi's interest in a tacit dimension to knowledge. Stemming directly from the introduction of computing machinery, his earliest writings addressing skill were formulated in

Skip to main content

response to “The Hypothesis of Cybernetics.” [41] Published in 1952 as part of a series of articles “on the question of whether machines can be said to think,” Polanyi’s succinct intervention subtly reframed the issue raised just two years earlier in Alan Turing’s provocations on the equivalence between “Computing Machinery and Intelligence” (1950). [42] Rather than directly responding to the possibility of thinking machines, Polanyi posed “the question of whether the operations of a formalised deductive system might conceivably be equivalent to the operations of the mind.” His terminological shift away from what others had called “psychological” factors (i.e. thinking) to an emphasis on “unformalised” elements redirected the challenge raised by computing towards “ordinary speech,” the exercise of which he considered to be grounded in the competency of “proper use”:

> No undefined term can be introduced unless its use is first explained by ordinary speech or demonstrated by examples. An undefined term is a sign indicating the proper use which is to be made of it. The acceptance of an undefined term implies, therefore, that we believe that we know its proper use though this is not to be formally described. This proper use is a skill of which we declare ourselves to be possessed.

Like Ryle, Polanyi did not seek to test claims of equivalence but to clarify what he considered a “logical fallacy,” targeting the realm of the nonsensical rather than the false. Whether or not any implementation in machinery could pass Turing’s test was for him besides the point; of sole relevance was the processing of a symbolic system, and he maintained “that a formal system of symbols and operations functions as a deductive system only by virtue of unformalised supplements.”

Computer science may most readily be associated with the construction of formalisms like programming languages. However, a persistent strand that finds its most overt expression as “procedural epistemology” instead addresses the cultural techniques of this supplementary excess. Whether in Minsky’s preference for “Poorly-Understood and Sloppily Formulated Ideas,” or Sussman’s claim that “it wouldn’t be enough to memorize all of the facts in the plumber’s handbook,” concern is displaced from what is explicitly articulated to techniques beyond formalization; these statements about statement-making address the threshold operations that process the emergence of the symbolic, the limits at which “unformalised supplements” may be transformed into a “formal system.”

---

1. Harold Abelson and Gerald Jay Sussman, with Julie Sussman, *Structure and Interpretation of Computer Programs*” (Cambridge, MA: MIT Press, 1985), xvi; hereafter abbreviated *SICP*. ↩
2. Abelson, *SICP*, xvi. ↩
3. Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi, “The structure and interpretation of the computer science curriculum,” *Journal of Functional Programming* 14.4 (July 2004), 365. ↩
4. For an account that contests the merits of Lisp-like languages in this context, cf. Philip Wadler, “A critique of Abelson and Sussman, or, Why calculating is better than scheming,” *ACM SIGPLAN Notices* 22, no. 3 (March 1987), 83-94. Wadler argues: “Some people may wish to dismiss many of the issues raised in this paper as being ‘just syntax.’ It is true that much debate over syntax is of little value. But it is also true that a good choice of notation can greatly aid learning and thought, and a poor choice can hinder it” (85). ↩

5. See Paul Ceruzzi, “Electronics Technology and Computer Science, 1940-1975: A Coevolution,” *Annals in the History of Computing* 10, no. 4 (1989), 257-275. ↩
6. Wadler, “Critique of Abelson and Sussman,” 93. ↩
7. Wendy Hui Kyong Chun,*Programmed Visions: Software and Memory* (Cambridge, MA: MIT Press, 2011), 51; N. Katherine Hayles,*My Mother Was a Computer: Digital Subjects and Literary Texts* (Chicago; London: University of Chicago Press, 2005), 49; Alexander Galloway, “Language Wants To Be Overlooked: On Software and Ideology,” *Journal of Visual Culture* 5, no. 3 (December 2006), 325-326. Chun’s description of “source code as fetish” raises the point of readability: “source code must be able to function, even if it does not function – that is, even if it is never executed.... When programming, one must be able to read one’s own program – to follow its logic and to predict its outcome, whether or not this outcome coincides with one’s prediction” (*Programmed Visions*, 51). ↩
8. Donald Knuth, “Literate Programming,” *The Computer Journal* 27, issue 2 (January 1984), 97-111. ↩
9. Edsger W. Dijkstra, *A Discipline of Programming* (Englewood Cliffs, NJ: Prentice-Hall, 1976), 1. ↩
10. Friedrich Kittler, “Code,” in *Software Studies: A Lexicon*, ed. Matthew Fuller, trans. Tom Morrison with Florian Cramer (Cambridge, MA: MIT Press, 2008), 46. ↩
11. Kittler writes: “Maternal instruction, in its positivity, was the input component of elementary acculturation techniques. Around 1800 a new type of book began to appear, one that delegated to mothers first the physical and mental education of their children, then their alphabetization. ... The titles speak for themselves. They leave little doubt about the identity of the instructor recommended; they emphasize that it is only by conferring elementary acculturation techniques on mothers that the Self of this identity has been found.” Friedrich Kittler, *Discourse Networks 1800/1900*, trans. Michael Meteer, with Chris Cullen (Stanford, CA: Stanford University Press, 1985), 27-53, here 27-28. This usage puts a polemic twist on the scope of the cultured liberal arts and their disavowed relationship towards technical competency. As Bernard Dionysius Geoghegan has noted, “according to [Erhard] Schüttpelz, Kittler encountered the term as a student and instructor at the University of Freiburg in the late 1970s and early 1980s, when the term *Kulturtechniken* was resurfacing in German as a designation for competencies in reading, writing, and arithmetic. This definition recalled the 18th- and 19th-century definition of culture as liberal arts.” See Geoghegan, “After Kittler: On the Cultural Techniques of Recent German Media Theory,” *Theory, Culture & Society* 30, no. 6 (2013), 76. ↩
12. Kittler, *Discourse Networks*, 28. ↩
13. See Bernard Siegert, *Cultural Techniques: Grids, Filters, Doors, and Other Articulations of the Real*, trans. Geoffrey Winthrop-Young (New York: Fordham University Press, 2015). ↩
14. Michel Foucault, *The Archaeology of Knowledge*, trans. A.M. Sheridan Smith (London; New York: Routledge, 2002), 161. ↩
15. See Ceruzzi, “Electronics Technology and Computer Science,” 265. ↩
16. John Backus, “Programming in America in the 1950s – Some Personal Impressions,” in *A History of Computing in the Twentieth Century: A Collection of Essays*, eds. N. Metropolis, J. Howlett, and Gian-Carlo Rota (New York; London: Academic Press, 1980), 126. ↩
17. Nathan Ensmenger, *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise* (Cambridge, MA: MIT Press, 2010), 43. ↩
18. Backus, “Programming in America,” 126. Backus himself worked on technologies to formalize these processes, including the programming language Fortran and a notation for precise description of the syntax of such languages, the Backus-Naur Form. See further, for Backus’s commentary on Fortran and his notation system, 130-133. ↩
19. Edsger W. Dijkstra, “EWD480: Craftsman or Scientist?” (1975), in *Selected Writings on Computing: A Personal Perspective*, ed. David Gries (New York: Springer-Verlag, 1982), 105. ↩
20. Edsger W. Dijkstra, “EWD1305: Answers to questions from students of Software Engineering” (unpublished manuscript, 28 November 2000). ↩
21. Cf. Ensmenger, *The Computer Boys Take Over*, especially “Chapter 5: The Rise of Computer Science,” 111-136; and, on the role of linguistic conceptions in establishing ideals such as machine-independence, David Nofre, Mark Priestley, and Gerard Alberts, “When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950-

1960," *Technology and Culture* 55.1 (2014), 40-75. In reflections elsewhere especially pertinent to the direction taken here, Priestley notes: "my practical experience of programming had led me to understand it as a very different type of activity from the formal manipulations of logical proof." See Mark Priestley, *A Science of Operations: Machines, Logic and the Invention of Programming* (London: Springer-Verlag, 2011), v. Wendy Chun's characterization of writing code as a "crafty" practice helpfully highlights the "possibility of deviousness" implied in the concept of craft, shifting the emphasis onto the role of "belief" in computing: "Code does not always or automatically do what it says, but it does so in a crafty, speculative manner in which meaning or action are both created" (Chun, *Programmed Visions*, 24). My argument covers similar ground but stresses the historicity in formations of epistemic legitimacy rather than an ontology of code. ↩

22. Dijkstra, "Craftsman or Scientist?," 106. ↩
23. Marvin Minsky, "Why Programming Is a Good Medium for Expressing Poorly-Understood and Sloppily Formulated Ideas," in *Design and Planning* 2, eds. Martin Krampen and Peter Seitz (New York: Hastings House, 1967), 121; cited in Abelson and Sussan, *SICP*, xv. ↩
24. Minsky, "Why Programming is a Good Medium," 117 and 121. ↩
25. Gerald Jay Sussman, "A Computational Model of Skill Acquisition," (PhD diss., MIT, 1973), 5-6. ↩
26. Tim Ingold, *The Perception of the Environment: Essays on Livelihood, dwelling and skill* (New York; London: Routledge, 2000), 291. ↩
27. Richard Sennett, *The Craftsman* (Yale University Press, 2008), 7, 9, and 179. ↩
28. Lucy A. Suchman, *Plans and Situated Actions: The problem of human-machine communication* (Palo Alto, CA: Xerox Corporation, Palo Alto Research Centers, 1985), 36-37. In an updated edition, Suchman offers reflections on the reception and readings of the original text. See Lucy A. Suchman, Human-Machine Reconfigurations: Plans and Situated Actions, 2nd Edition (Cambridge, UK: Cambridge University Press, 2007). ↩
29. Suchman, *Plans and Situated Actions*, 37-38. ↩
30. Bernard Stiegler, *Technics and Time, 1: The Fault of Epimetheus*, trans. Richard Beardsworth and George Collins (Stanford, CA: Stanford University Press, 1998), 1. ↩
31. Gilbert Ryle, *The Concept of Mind* (London: Routledge, [1949] 2009), 14-48. ↩
32. Michael Polanyi, *Personal Knowledge: Towards a Post-Critical Philosophy* (London: Routledge, [1958] 1962), esp. "Skills," 51-68. ↩
33. Ryle, *The Concept of Mind*, 18 and 46. ↩
34. Suchman, *Plans and Situated Action*, 3. Cf. Ryle's "Introduction" to *The Concept of Mind*, lix-lxi. ↩
35. Thomas Gladwin, "Culture and Logical Process," in *Explorations in Cultural Anthropology: Essays in Honor of George Peter Murdock*, ed. Ward H. Goodenough (New York; San Francisco; Toronto; London: McGraw-Hill Book Company, 1964), 173. ↩
36. Suchman, *Plans and Situated Action*, p. 1. An addition in the updated edition states even more clearly: "we all act like the Trukese, however much some of us may talk like Europeans" (Suchman, *Human-Machine Reconfigurations*, 26). ↩
37. Thomas Gladwin, *East Is a Big Bird: Navigation & Logic on Puluwat Atoll* (Cambridge, MA: Harvard University Press, 1970), 232. ↩
38. Ingold, *Perception of the Environment*, 289. ↩
39. On the history of information techniques as a "government machine," see Jon Agar, *The Government Machine: A Revolutionary History of the Computer* (Cambridge, MA: MIT Press, 2003). ↩
40. See Peter Galison, "Computer Simulations and the Trading Zone," in Peter Galison and David J. Stump (eds.), *The Disunity of Science: Boundaries, Contexts, Power* (Stanford, CA: Stanford University Press, 1996), 118-157; and Paul N. Edwards, *A Vast Machine: Computer Models, Climate Data, and the Politics of Global Warming* (Cambridge, MA: MIT Press, 2013). See also, Gabriele Gramelsberger (ed.), *From Science to Computational Sciences: Studies in the History of Computing and its Influence on Today's Sciences* (Zurich: Diaphanes, 2011). ↩

Skip to main content

41. See Michael Polanyi, “The Hypothesis of Cybernetics,” *The British Journal for the Philosophy of Science* 2.8 (Feb 1952), 312-315. ↩
42. See A.M. Turing, “Computing Machinery and Intelligence,” *Mind* 49.236 (October 1950), 433-460. ↩