THINK PIECE
Annals of the History of Computing

[preprint version: *10 October 2021*][1]

Addressing the Question "What is a Program Text?" via Turing Scholarship

Edgar G. Daylight
Siegen University & Lille University[1]

For decades, a range of disciplines (from engineering and computer science to philosophy, law, and history) have seen debates over the nature of computer programs. Various scholars—including Paul Ceruzzi, Timothy Colburn, Gerardo Con Diaz, Liesbeth De Mol, Nathan Ensmenger, Thomas Haigh, and Matti Tedre—have advanced a wide array of views on the matter, debating whether programs are machines, texts, algorithms, hybrids, or even technologies in the first place.[2] In this Think Piece, I use notions of "law" developed by Arthur Eddington and other contemporaries of Alan Turing to explore what a program *text* (as opposed to a *computer* program) is.

In the 1920s, Arthur Stanley Eddington, the English astronomer and philosophical enthusiast, proposed a distinction between two kinds of laws: "human or governing laws" on the one hand and "scientific or natural laws"—as found in geometry, mechanics, and physics—on the other hand.[3] "In human affairs," he wrote, a "law" means "a rule, fortified perhaps by incentives or penalties, which may be kept or broken," while in science it means "a rule which is never broken; we suppose that there is something in the constitution of things which makes its non-fulfilment an impossibility."[4] For Eddington, this distinction was consistent with his credo that not all kinds of knowledge of the real world are "ultimately reducible to mathematical equations," nor thus to natural laws. In this regard he disagreed with many of his colleagues who kept their trust in "the universal dominance of scientific law."[5]

Although Eddington became increasingly perceived as a dilettantisch philosopher during the 1920s-1930s, both by contemporary scientists and specialist philosophers, historians of computer science might well want to

[1] The present document has subsequently been technically edited, which has led to the official publication.

have a closer look at his writings in which he discussed the philosophical and religious implications of general relativity theory and quantum mechanics to a broad audience.[6] For that audience included another philosophically inclined mind, the young Alan Turing, who would become one of Eddington's students at Cambridge University.[7] As Andrew Hodges suggests in a philosophical account of Turing, there is reason to believe, that, besides Eddington's scientific outlook on an uncertain world, parts of his philosophical doctrine rubbed off on Turing as well.[8] The following extract, from an essay draft (entitled "The Nature of Spirit") written by Turing around 1932, reveals his closeness with both Eddington's scientific work and with mysticism:

> It used to be supposed in Science that if everything was known about the Universe at any particular moment then we can predict what it will be through all the future. … More modern science however has come to the conclusion that … we are quite unable to know the exact state …
>
> As McTaggart shews[,] matter is meaningless in the absence of spirit … Personally I think that spirit is really eternally connected with matter but certainly not always by the same kind of body. …[9]

The extent to which Eddington and the late John McTaggart (mentioned in the previous passage) influenced Turing is part of my on-going research. Similar questions hold regarding Ludwig Wittgenstein (some of whose classes Turing attended in 1939) and Dorothy Sayers whose 1941 book, *The Mind of the Maker,* Turing read during the war.[10] In her book, Sayers engaged with Eddington's 1939 work *The Philosophy of Physical Science* and elaborated the afore-mentioned distinction between governing laws and natural laws—a topic which, in my reading, was of central concern in the 1939 exchanges between Wittgenstein and Turing.[11]

All this to motivate, as a Turing scholar, my choice of historical actors and my inquiries regarding the status of laws, symbolic knowledge, rule following, the so-called foundations of mathematics (before the Second World War) and of computer science (after the war). These explorations naturally result in fascinating detours which are worth sharing with fellow scholars. One such possible detour, presented in the sequel to this introduction, concerns the notions of "law" as developed by Eddington, Sayers, and Wittgenstein, which I shall use to explore what a *program text* meant in the 1980s.

Three Categories

In *The Mind of the Maker,* Sayers reinforced Eddington's distinction between "an arbitrary code of behavior based on a consensus of human opinion" and "a statement of unalterable fact about the nature of the universe."[12] Sayers subsequently responded to Eddington's dichotomy by highlighting a third kind of law in her book: a statement of historical fact. One of her examples was Grimm's Law, which "may be defined as the statements of certain phonetic facts which happen invariably unless they are interfered with by other facts."[13] A modern example is Moore's Law, which captures a historical trend, stating that the number of transistors in a dense integrated circuit, doubles about every two years.[14]

As stated so far, these categories of Eddington and Sayers are:

1. Governing laws, which are prescriptive but not *per se* descriptive; they are recommendations that are authoritatively put forward, but they are not necessarily followed in the real world.
2. Natural laws, which are both prescriptive and descriptive: the logico-mathematical symbols express what will happen physically and, vice versa, *what does happen is captured in the symbolism*.
3. Historical trends, which are primarily descriptive and secondarily (if at all) prescriptive.

My working hypothesis is, that, in 1939, Eddington, Turing, and Wittgenstein each placed symbolic logic in a different category. Eddington positioned logic in the first category of governing laws.[15] In conversation with Wittgenstein, Turing situated symbolic logic in category 2, in conformity with Russell's intellectual position.[16] Wittgenstein protested, insisting that it belongs in category 3. To clarify Wittgenstein's stance, I now propose to extend Sayers's category 3 to include maps and tools as well.

Historical trends, maps, and tools are objects—some abstract and others concrete—which are primarily *descriptive* in a broad sense of the word.[17] For example, Moore's Law is used daily *as a map, a tool,* by circuit designers and managers at microelectronics centers to predict the fabrication cost of a next-generation chip. As a second example, a city map informs me how I could travel to reach my destination. Neither Moore's Law nor the city map dictate how to get a specific job done. In each case, the prescriptive component is of secondary importance, if

present at all. Wittgenstein held the view—pace Ray Monk's biography—that symbolic logic can only be used as a kind of *tool*, e.g., by an architect to design and construct a bridge.[18] The architect works with a calculus (and today with a software tool); that is, with fragments of symbolic logic. When she encounters an inconsistency in the calculus, she is free to resort to another calculational technique to complete her design. The calculus, like a tool, *does not dictate* how she must proceed.

In this Think Piece I maintain that these three notions of what symbolic logic entails also pertain to a *program text* written in a programming language. Making this connection corrects what has become a widespread assumption in the philosophy of computer science as found in the Stanford Encyclopedia of Philosophy's entry, *The Philosophy of Computer Science*, by Nicola Angius, Giuseppe Primiero, and Raymond Turner (herein after APT).[19] In my reading, the APT authors implicitly advocate the view that well-chosen symbols in a suitable formalism (e.g., a program text in a programming language) are to be interpreted as a kind of natural law. This aligns with computer scientist Tony Hoare's 1969 view on the matter, which implicitly hinges on a Russellian isomorphism between symbols in logic and objects in physics—an assumption which has frequently been overlooked in academic computing circles until James Fetzer's philosophical critique appeared in the *Communications of the ACM*, a flagship periodical in computer science.[20] Donald MacKenzie provides a compelling socio-historical account of these intellectual developments in his *Mechanizing Proof*.[21]

I coin the isomorphism between symbols and physical objects "Russellian" after Bertrand Russell, the analytic philosopher renowned for championing logicism in his 1903 book *The Principles of Mathematics*, i.e., the idea that "all Mathematics is Symbolic Logic," a so-called "fact" which he took to be "one of the greatest discoveries of our age."[22] He wanted to "discover a logically ideal language … that will exhibit the nature of the world in such a way that we will not be misled by the accidental, imprecise surface structure of natural language."[23] A decade later, the young Ludwig Wittgenstein would partly join Russell on this quest for rationality. Fast forward to the 1930s and we see Wittgenstein in his forties teaching almost the complete opposite philosophy at Cambridge University, much to Russell's dismay.[24]

What is a Program Text?

The three categories can now come into play when analyzing a 'program text,' written in a modern programming language:[25]

1. An engineering perspective on 'program text' means that the text expresses what the follow-up compilation and runtime processes *need* to accomplish and that the actual program execution can deviate from what the text prescribes. Hence, the program text exemplifies a governing law for the software engineer.

2. The same program text is to be interpreted as a kind of natural law for the Russellian computer scientist. The program text expresses what the compilation and runtime processes need to accomplish *and* the actual program execution can, in principle, *fully* abide by these stipulations. To use Hoare's famous 1969 words:

   > Computer programming is an exact science in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely deductive reasoning.[26]

   The worlds of symbolic and physical computations seem isomorphic to one another; no distinction is in order between the prescriptive and the descriptive.

3. The same program text exemplifies a map for the engineer who has reverse engineered legacy software into the text at hand. The obtained description is a digital approximation of the behavior of the software in continuous physics. Engineers use the description as a map, as a tool, to find out what the legacy software (say, a COBOL program) accomplishes in the real world.

Consider the third example: Hoare's 1969 article and the APT authors' 2021 encyclopedia entry seem to imply that the program text can in principle *fully* capture the physical computation (of the legacy software). They hold the view that *all essentials of physical (continuous) computation can be captured with a symbolic (digital) representation*. This, then, is my general complaint, despite the otherwise excellent survey provided by the APT authors. According to their philosophy, the map (3.) can, in principle, also serve as a governing law (1.), and hence is akin to a natural law (2.).

Hoare's research agenda has been criticized, not only by the philosopher James Fetzer, but also by his colleagues in computing circles. I briefly

focus on the views of David Parnas and Peter Naur in the 1980s. I situate the no-nonsense engineer Parnas in category 1 and I place Naur in Wittgenstein's camp (category 3).

Parnas and Naur had in common that they both regarded many computer scientists as researchers who frequently conflate the mathematical results of their model with the properties of their modeled artefact.[27] This observation aligns with the historical account in MacKenzie's *Mechanizing Proof* and lends further credence to Phil Agre's 1997 forceful words: "the conflation of representation and reality is a common and significant feature of the whole computer science literature."[28] The APT authors, however, are imperceptive to such matters and implicitly side with Hoare.

Naur, however, went a step further than Parnas in critizing Russellian computer scientists. While both Hoare and Parnas took programming to be primarily about producing texts, Naur perceived programming rather as an *activity* by which the programmer achieves a certain kind of insight, a *theory*, of the matter at hand—something which cannot be completely expressed with symbols.[29] As he told me in person, "program development is about building up a certain understanding, a theory, it's not about creating a program text."[30] In response, I asked him what a software company should do when the main programmers leave the company, i.e., when "the program is dead."[31] Naur replied that "it would be better that the new programmers" of the company at hand "start building their own program," i.e., that they discard the dead program and start from scratch.[32] Although the APT authors do bring up Naur's theory building perspective on computer programming in their encyclopedia entry, they do not convey the fundamental difference in philosophical outlook between Hoare and Naur (i.e., between Russell and the later Wittgenstein), let alone between Parnas and Naur.

A worm's-eye view on the Parnas-Hoare-Naur triangle amounts to making the following (final) observations. According to Parnas, we can only get a rational and complete description of the software design process *after* the product at hand has been engineered, not *before* as some hard-core formal methods people in Hoare's intellectual circle advertised in the 1970s.[33] According to Naur, however, we cannot even get a complete and rational description a posteriori. The programmer's theory is center stage, while a program text is merely a by-product and far from perfect.[34] "Science," Naur told me, "has nothing to do with logic or truth. It is merely a matter of description. Descriptions are not true, never, they are more or less adequate. That's the best they can offer. They are useful."[35]

Historians of philosophy will appreciate a clear connection between Naur in the 1980s and Wittgenstein in the 1940s (cf. category 3).

All this to urge the APT authors to step away from a purely Russellian account of our computer-equipped world, or to simply inject the adjective "analytical" in their grand title (*The Philosophy of Computer Science*). Doing so would allow Eddingtonian and Wittgensteinian scholars to have their say too, in what is more aptly called: *Philosophies of Computer Science*.

---

[1] The author, a.k.a. Karel Van Oudheusden, was financed by SFB 1187 "Medien der Kooperation" (Siegen University) and by ANR-17-CE38-003-01 "PROGRAMme" (Lille University). He is grateful for receiving detailed feedback on drafts of this text from Nick Wiggershaus, Gerardo Con Diaz, Erhard Schüttpelz, and Liesbeth De Mol. He also thanks several other PROGRAMme members and Michael Jackson, Edward A. Lee, and Kjeld Schmidt for their comments.

[2] Paul Ceruzzi, *Computing, a Concise History* (Cambridge, MA: MIT Press, 2012); Timothy Colburn, *Philosophy and Computer Science* (New York: M.E. Sharpe, 2000); Gerardo Con Diaz, *Software Rights: How Patent Law Transformed Software Development in America* (New Haven & London: Yale University Press, 2019); Liesbeth De Mol, Maarten Bullynck, "Roots of 'program' revisited," *Communications of the ACM* 64, no. 4 (April 2021): 35-37; Nathan Ensmenger, *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise* (Cambridge, MA: MIT Press, 2012); Thomas Haigh (ed.), *Exploring the Early Digital* (Zurich: Springer International Publishing, 2019); Matti Tedre, *The Science of Computing: Shaping a Discipline* (London: CRC Press, 2015).

[3] Eddington explicitly acknowledged his very limited reading of philosophy on multiple occasions; see, e.g., Arthur Eddington, *The Philosophy of Physical Science* (Cambridge: At the University Press, 1939), 155; quotations come from: Arthur Eddington. *Science and the Unseen World: Swarthmore Lecture 1929* (Birmingham: Quaker books, 2007), Chapter V.

[4] Arthur Eddington, *Science and the Unseen World: Swarthmore Lecture 1929* (Birmingham: Quaker books, 2007), 33.

[5] Arthur Eddington, *Science and the Unseen World: Swarthmore Lecture 1929* (Birmingham: Quaker books, 2007), 30.

[6] Herbert Dingle, *The Sources of Eddington's Philosophy* (Cambridge: At the University Press, 1954), 40-41; CEM Joad, *Philosophical Aspects Of Modern Science* (London: George Allen & Unwin Lt, 1932), 37-38.

[7] Andrew Hodges, *Alan Turing: The Enigma* (London: Burnett Books, 1983), 51, 87.

[8] Andrew Hodges, "Turing," in *The Great Philosophers: From Socrates to Turing*, ed. Ray Monk, Frederic Raphael (London: Phoenix, 2001), 497, 541.

[9] The entire quote appears in: Andrew Hodges, *Alan Turing: The Enigma* (London: Burnett Books, 1983), 63-64; The draft essay itself is archived under Turing catalog number AMT/C/29, *www.turingarchive.org/browse.php/C/29*

[10] Ray Monk, *Ludwig Wittgenstein: The Duty of Genius* (London: Vintage Books, 1991), Chapter 20; Dorothy Sayers, *The Mind of the Maker* (New York: Harcourt, 1941); republished (New York: HarperCollins Publishers, 1979); Andrew Hodges, *Alan Turing: The Enigma* (London: Burnett Books, 1983), 211.

[11] Arthur Eddington, *The Philosophy of Physical Science* (Cambridge: At the University Press, 1939); Cora Diamond (ed.), *Wittgenstein's Lectures on the Foundations of Mathematics* (Ithaca: Cornell University Press, 1976).

[12] Dorothy Sayers, *The Mind of the Maker* (New York: Harcourt, 1941); republished (New York: HarperCollins Publishers, 1979), 8.

[13] Dorothy Sayers, *The Mind of the Maker* (New York: Harcourt, 1941); republished (New York: HarperCollins Publishers, 1979), 6.

[14] Edward A. Lee notes that "most industry observers seem to agree that as of 2015, [Moore's law] has finally significantly slowed," cf. Edward A. Lee, *Plato and the Nerd: The Creative Partnership of Humans and Technology* (Cambridge, MA: MIT Press, 2017), 105.

[15] Eddington had spoken the words “The laws of logic do not prescribe the way our minds think; they prescribe the way our minds ought to think” ten years earlier; cf. Arthur Eddington, *Science and the Unseen World: Swarthmore Lecture 1929* (Birmingham: Quaker books, 2007), 33. I take his 1929 exposition to be a build-up to his full-fledged 1939 philosophical account, cf. Eddington, *The Philosophy of Physical Science* (Cambridge: At the University Press, 1939); Herbert Dingle, *The Sources of Eddington’s Philosophy* (Cambridge: At the University Press, 1954), 46.
[16] Whether Turing shared Russell’s view or merely intended to protect it from Wittgenstein’s intellectual charge is yet another matter. Regarding the dialogue itself, I rely on: Cora Diamond (ed.), *Wittgenstein’s Lectures on the Foundations of Mathematics* (Ithaca: Cornell University Press, 1976).
[17] Just as a category of yellow objects contains *both* abstract and concrete objects. To my surprise, some readers (who do embrace dualism) take issue with this generalization. But rest assured, my Think Piece does not fundamentally hinge upon it.
[18] Ray Monk, *Ludwig Wittgenstein: The Duty of Genius* (London: Vintage Books, 1991), Chapter 20.
[19] Nicola Angius, Giuseppe Primiero, and Raymond Turner, "The Philosophy of Computer Science," *The Stanford Encyclopedia of Philosophy* (Spring 2021 Edition), Edward N. Zalta (ed.), URL = <https://plato.stanford.edu/archives/spr2021/entries/computer-science/>.
[20] See also Hoare’s personal comments to me: Edgar Daylight, “From Mathematical Logic to Programming-Language Semantics — a Discussion with Tony Hoare,” *Journal of Logic and Computation* 25, no. 4 (2015): 1101; James Fetzer, “Program Verification: The Very Idea,” *Communications of the ACM* 31, no. 9 (1988): 1048-1063.
[21] Donald MacKenzie, *Mechanizing Proof: Computing, Risk, and Trust* (Cambridge, MA: MIT Press, 2004), Chapter 6.
[22] Bertrand Russell, *The Principles of Mathematics* (Cambridge: At the University Press, 1903); Bertrand Russell, *The Principles of Mathematics* (Oxfordshire: Routledge, 2009): 5.
[23] Andrew David Irvine, "Bertrand Russell," *The Stanford Encyclopedia of Philosophy* (Summer 2017 Edition), Edward N. Zalta (ed.), URL = <https://plato.stanford.edu/archives/sum2017/entries/russell/>.
[24] Ray Monk, *Wittgenstein* (London: Granta Books, 2005): 68.
[25] Any so-called “machine-independent” or “universal” or “Algol-like” programming language will do, for the post-war quest for such a utopian language retrospectively mirrors nothing less than pre-war Russellian logicism. In connection to these statements see, e.g., David Nofre, Mark Priestley, Gerard Alberts, “When Technology Became Language: The Origins of the Linguistic Conception of Computer Programming, 1950-1960,” *Technology and Culture* 55, no. 1 (2014): 40-75; Gerard Alberts, Edgar Daylight, “Universality versus Locality: The Amsterdam Style of Algol Implementation,” *IEEE Annals of the History of Computing* 36, no. 4 (2014): 52-63.
[26] Tony Hoare, “An Axiomatic Basis for Computer Programming,” C*ommunications of the ACM* 12, no.10 (1969): 576.
[27] In my reading, Parnas and Naur take the joint position, that, it is a metaphysical superposition to believe—cf. Russell’s isomorphism—that items of constructed models are inherent in their modellees. A theoretical incomputability result (such as the Halting Problem) has immediate implications in physical computing according to Hoare, but not for Parnas, Naur, and others. See: Edgar Daylight, *Pluralism in Software Engineering: Turing Award Winner Peter Naur Explains* (Heverlee: Lonely Scholar, 2011), 83; Edgar Daylight, “The Halting Problem and Security's Language Theoretic Approach: Praise and Criticism from a Technical Historian,” *Computability* 10, no. 2 (2021): Section 1.3.
[28] Donald MacKenzie, *Mechanizing Proof: Computing, Risk, and Trust* (Cambridge, MA: MIT Press, 2004), Chapter 6; quoted from: Philip E. Agre, “Beyond the Mirror World: Privacy and the Representational Practice of Computing,” in *Technology and Privacy: The New Landscape*, ed. Philip E. Agre, Marc Rotenberg (Cambridge: MIT Press, 1997), Chapter 1; Edgar Daylight, “The Halting Problem and Security's Language Theoretic Approach: Praise and Criticism from a Technical Historian,” *Computability* 10, no. 2 (2021): Section 2.2.
[29] Peter Naur, “Programming as Theory Building,” *Microprocessing and Microprogramming* 15, no. 5 (1985): 253-261.
[30] Edgar Daylight, *Pluralism in Software Engineering: Turing Award Winner Peter Naur Explains* (Heverlee: Lonely Scholar, 2011), 67.
[31] Edgar Daylight, *Pluralism in Software Engineering: Turing Award Winner Peter Naur Explains* (Heverlee: Lonely Scholar, 2011), 67.
[32] Edgar Daylight, *Pluralism in Software Engineering: Turing Award Winner Peter Naur Explains* (Heverlee: Lonely Scholar, 2011), 67.
[33] David Parnas, Paul Clements, “A Rational Design Process: How and Why to Fake It,” *IEEE Transactions on Software Engineering* 12, no. 2 (1986): 251-257; Edsger W. Dijkstra, *A Discipline of Programming* (Englewood Cliffs, N.J.: Prentice-Hall, 1976).

[34]Peter Naur, “Programming as Theory Building,” *Microprocessing and Microprogramming* 15, no. 5 (1985): 253-261.
[35] Edgar Daylight, *Pluralism in Software Engineering: Turing Award Winner Peter Naur Explains* (Heverlee: Lonely Scholar, 2011), 86.