Chapter 7

"Difficult Things are Difficult to Describe": The Role of Formal Semantics in European Computer Science, 1960–1980

Troy Kaighin Astarte

In 1970, Friedrich Bauer sketched a history of computation from abacuses to ALGOL 68 which he presented as a pinnacle of achievement in computing.[1] ALGOL 68 was the culmination of the ten-year international ALGOL project, "the seed around which computer science began to crystalize as an academic discipline".[2] A key aspect of the project was the use of formalism in the language description; formalism meant abstractions, fixed structures, and rule-based reasoning borrowed from formal logic. ALGOL 68's predecessor ALGOL 60 had a formalised syntax, and the ALGOL Committee had unanimously resolved in 1964 that the new language must be "defined formally and as strictly as possible in a meta-language".[3]

Since the late 1950s, a certain community of influential computing researchers began to view formal specification of programming languages as a crucial part of research into computation, and central to the practice of language design and implementation. Many in this community worked on formal semantics during the 1960s, but by the end of the 1970s most had moved off the subject. This mirrored a general trend in computing research away from languages and towards programs or programming techniques.[4] Part of this trend was the rising interest in program verification, a topic which is better known to historians than semantics.[5] Indeed, the two

---

[1] Friedrich L. Bauer, "From Scientific Computation to Computer Science," in *The Skyline of Information Processing: Proceedings of the Tenth Anniversary Celebration of the IFIP*, ed. Heinz Zemanek (North-Holland, 1972).

[2] Thomas Haigh and Paul E. Ceruzzi, *A New History of Modern Computing*, (MIT Press 2021), 43.

[3] R. E. Utman, "Minutes of the 3rd Meeting of IFIP WG 2.1", March 1964, 19–20, Ershov Archive, http://ershov.iis.nsk.su/en/node/778087.

[4] Peter Wegner, "Research Paradigms in Computer Science," in *Proceedings of the 2nd International Conference on Software Engineering* (IEEE Computer Society Press, 1976), 322–30.

most-cited works in verification both show a concern for semantics.[6] The study of formal semantics established an intellectual agenda which carried through into work on verification.

The history of computing shows the programming community in the late 1950s grappling with their work, which was turning increasingly professional, scientific, and impactful through its embedding in society. Programming, a nebulous practice somewhere between material and abstract, cried out for understanding: an ontological gap yawned. The opportunity to bridge the gap with tools and agendas from mathematics and especially formal logic also provided legitimisation for the nascent 'computer science,' allowing it to be rooted in, but distinct, from existing intellectual traditions. [7]

This chapter explores the context of formal semantics and situates the work within the history of computer science, discussing how and why the community involved changed their focus to program correctness. The inability of formal language semanticists to embody their work in broader programming practices gives us clues to why formal semantics declined. An immediate reason was expressed in Hoare's remark "difficult things are difficult to describe".[8] Programming languages of the sort being developed in the 1960s contained many complex, interacting features that challenged easy abstractions; the same reasons made formal description embodiments in implementations difficult. More subtly, the community's strongly held belief that formal specification was a prerequisite for producing correct and reliable programs was

---

[5] See, for example, Donald MacKenzie, *Mechanizing Proof: Computing, Risk, and Trust* (MIT Press, 2001), and Matti Tedre, *The Science of Computing: Shaping a Discipline* (Chapman & Hall, 2014).

[6] R. W. Floyd, "Assigning Meanings to Programs," in *Mathematical Aspects of Computer Science*, ed. J. T. Schwartz, vol. 19, Proc. Of Symposia in Applied Mathematics (American Mathematical Society, 1967), 19–32; C. A. R. Hoare, "An Axiomatic Basis for Computer Programming," *Communications of the ACM* 12, no. 10 (1969): 576–80.

[7] Mark Priestley, *A Science of Operations: Machines, Logic and the Invention of Programming* (Springer Science & Business Media, 2011), 231. "Bridging" was coined in Andrew Pickering, *The Mangle of Practice: Time, Agency, and Science* (University of Chicago Press, 1995).

[8] C. A. R. Hoare, in Kurt Walk, "Minutes of the 3rd Meeting of IFIP WG 2.2 on Formal Language Description Languages," April 1969.

challenged by industry's *de facto* policy that 'good enough' programs could be produced without the lengthy and costly processes many saw as preliminary at best.[9]

A crisis in programming

At a NATO-sponsored conference in 1968 a crisis was declared: software had become too important to society for the sloppiness of modern programming.[10] One solution was 'software engineering,' interpreted by some as "the reduction of programming to a field of applied mathematics".[11] This tension between "technological apocalypticism" and a genuine fear in the "fragility of infrastructure" mirrors and perhaps foreshadowed the later Y2K crisis.[12] Though the threat was not as immediate in the 1960s both crises served as motivators to action. Unlike the very real danger posed by Y2K, however, the software crisis may have been a retroactive construction by academics, especially Dijkstra,[13] to justify their own research agenda: mathematical abstractions for specifying and verifying programs.

One cause for concern was programming languages, called by Dijkstra the "basic tool" of computing.[14] Early programming codes had a direct correspondence to machine operations;

---

[9] Donald MacKenzie, "A View from the Sonnenbichl: On the Historical Sociology of Software and System Dependability," in *History of Computing: Software Issues*, ed. Ulf Hashagen, Reinhard Keil-Slawik, and Arthur Norberg (Springer-Verlag, 2002), 97–122.

[10] Peter Naur and Brian Randell, eds., "Software Engineering: Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968" (1969).

[11] Michael S Mahoney, "Computers and Mathematics: The Search for a Discipline of Computer Science," in *The Space of Mathematics: Philosophical, Epistemological, and Historical Explorations*, ed. Javier Echeverria, Andoni Ibarra, and Thomas Mormann (De Gruyter, 1992), 352.

[12] Zachary Loeb, "Waiting for Midnight: Risk Perception and the Millennium Bug," this volume.

[13] Thomas Haigh, "Assembling a Prehistory for Formal Methods: A Personal View," *Formal Aspects of Computing* 31, no. 6 (2019): 667.

[14] E. W. Dijkstra, "The Humble Programmer," *Communications of the ACM* 15, no. 10 (1972): 862.

programs were written to fit the peculiarities of specific machines.[15] The high-level languages of the 1950s, more abstract in their notation and automatically translated to run on machines, departed from this direct meaning. Dissociating program from machine was a consequence of machine-independent languages, but opened up the possibility of new ontologies of programs. If a program was written in terms of abstract entities and its execution was understood in terms of hardware function, could a truly believable correspondence between the two be established? One critical step was the creation of the ALGOL languages, intended from their genesis to be machine-independent. This made the definition documents the ultimate reference, but fears remained that documents would prove insufficient to solve language ambiguities and determine implementation correctness.[16] Decoupling languages from machines reified them into objects of study in their own right, and ALGOL became the keystone of European programming: a paradigm shift in thinking about programming.[17]

Formalism addressed the ontological problem of programming by creating an abstraction of machine operations, thereby subjecting it to the (then prevalent) logico-mathematical view of human thought.[18] The use of computers by, for example, Newell and Simon in the early 1950s to generate mathematical knowledge represented a new kind of epistemology; Babintseva problematises the way in which this "mathematical reasoning [was accepted] as the standard of human thinking.[19] This American view of computation was moulded and reshaped in the

---

[15] María Eloína Peláez Valdez, "A Gift from Pandora's Box: The Software Crisis." (PhD thesis, University of Edinburgh, 1988), 4.

[16] Troy K. Astarte and Cliff B. Jones, "Formal Semantics of ALGOL 60: Four Descriptions in Their Historical Context," in *Reflections on Programming Systems - Historical and Philosophical Aspects*, ed. Liesbeth De Mol and Giuseppe Primiero (Springer Philosophical Studies Series, 2018), 83–152.

[17] Priestley, *A Science of Operations*, 229.

[18] Stephanie Dick, "Of Models and Machines: Implementing Bounded Rationality," *Isis* 106, no. 3 (2015): 623–34.

[19] Ekaterina Babintseva, "From Pedagogical Computing to Knowledge-Engineering: The Origins and Applications of Lev Landa's Algo-Heuristic Theory," this volume.

European scientific environment to form a different path for theoretical computer science: program correctness and 'formal methods' rather than complexity theory.[20]

Exploring semantics

In the 1950s, John McCarthy developed LISP as part of the same mathematical reasoning effort as Newell and Simon.[21] McCarthy then realised LISP could also be used to form abstractions for generic language features: recursive functions manipulating a program state.[22] McCarthy wrote these functions using the notation (but not the conversion rules) of lambda calculus; he later admitted he did not properly understand the system.[23] Lambda calculus comprises a notation for mathematical functions, and a series of conversion rules for its terms. Subsequent research demonstrated its expressive power to be equivalent to Turing machines; this 'Church-Turing' thesis was a significant result in logic and mathematics. By borrowing lambda notation—if not its substance—McCarthy placed his work in that lineage.

This was deliberate: McCarthy wanted to create a 'mathematical science of computation.' While logicians and mathematicians such as Gödel and Turing had worked on computability problems in the 1930s, the embodiment of computation in the electronic machines of the 1940s and 1950s created a new problem: what exactly could these computers *do*? McCarthy's goal was find the basic notions of computation and then explore what deductions could be made from there. One way to gain insight into the way practitioners think about their work is to examine the

---

[20] Cliff B. Jones, "The Early Search for Tractable Ways of Reasoning about Programs," *IEEE Annals of the History of Computing* 25, no. 2 (2003): 26–49; Stephanie Dick, "Computer Science," in *A Companion to the History of American Science*, ed. Georgina M Montgomery and Mark A. Largent (John Wiley & Sons, 2015), 79–93.

[21] John McCarthy, "History of LISP," in *History of Programming Languages*, ed. Richard L. Wexelblat (Academic Press, 1981), 173–97.

[22] John McCarthy, "Towards a Mathematical Science of Computation," in *IFIP Congress*, vol. 62, 1962, 21–28.

[23] John McCarthy, "LISP—Notes on Its Past and Future—1980," in *Proceedings of the 1980 ACM Conference on LISP and Functional Programming*, 1980, v–viii.

historical models they choose to frame their own work.[24] McCarthy demonstrates this presented pedigree by referring to Kepler's laws of planetary motion being derivable from Newton's laws of general motion, and placing his own work in the same tradition.[25] This desire to 'scientise' computing lent a legitimacy to the nascent field and was embodied by the growing use of the term 'algorithm,' a word associated with the scientific method, Newton and Leibniz, and procedural practice.[26]

McCarthy's work was discovered by Peter Landin in 1958; Landin was impressed by LISP, particularly its abstract interpreting machine. However, he spotted the problems with McCarthy's lambdas, and wanted to see if he could "[tie] up some loose ends".[27] At this time, Landin was working for a consulting business run by Christopher Strachey—ostensibly developing an Autocode compiler for a Ferranti computer.[28] Drawing on his background as a mathematics graduate, Landin decided to try using lambda calculus as an "interlingua" and hoped a compiler would emerge from the semantics of this language.[29] By 1963, he published his lambda calculus approach to semantics: translate parts of a language into a "syntactically sugared" form of lambda notation, and then interpret this with an abstract computing machine.[30]

---

[24] Michael S Mahoney, "The History of Computing in the History of Technology," *Annals of the History of Computing* 10, no. 2 (1988): 113–25.

[25] Michael Sean Mahoney, "Software as Science—Science as Software," in *History of Computing: Software Issues*, ed. Ulf Hashagen, Reinhard Keil-Slawik, and Arthur Norberg (Springer-Verlag, 2002), 25–48.

[26] Liesbeth De Mol and Maarten Bullynck, "What's in a Name? Origins, Transpositions and Transformations of the Triptych Algorithm – Code – Program", this volume.

[27] Peter J. Landin, "Reminiscences," in *Program Verification and Semantics: The Early Work*, BCS Computer Conservation Society Seminar, Science Museum, London, UK, 5 June 2001, https://vimeo.com/8955127.

[28] Peter J Landin, "My Years with Strachey," *Higher-Order and Symbolic Computation* 13, no. 1 (2000): 75–76.

[29] Richard Bornat, "Peter Landin: A Computer Scientist Who Inspired a Generation, 5th June 1930 - 3rd June 2009," *Formal Aspects of Computing* 21, no. 5 (October 1, 2009): 393–95.

[30] Peter J. Landin, "The Mechanical Evaluation of Expressions," *The Computer Journal* 6, no. 4 (1964): 308–20.

Concurrently, Strachey was also becoming interested in programming languages. Having co-authored a paper with Maurice Wilkes criticising some aspects of ALGOL 60, he was invited by Wilkes to take part in developing a new language, CPL.[31] Strachey threw himself into the task, but became more interested in formalising the language's semantics than writing a compiler.[32] Landin's approach appealed to Strachey, who wrote with pride that he was sponsoring "the only work of its sort [language theory] being carried out anywhere (certainly anywhere in England)."[33]

In September 1964, McCarthy, Landin, and Strachey attended an International Federation for Information Processing (IFIP) working conference near Vienna, entitled *Formal Language Description Languages* (FLDL).[34] This conference brought together programmers, mathematicians, and linguists, allowing sharing of ideas and setting of a research agenda: namely, that programming languages were worthy objects of study, and that formalism provided the toolkit for that study.[35] It is no coincidence the ALGOL Committee (IFIP Working Group 2.1) decided in 1964 to fully formalise ALGOL 60's successor: their autumn meeting coincided with FLDL to draw from the conference's findings.[36]

One important outcome of FLDL was its influence on the new IBM Research Laboratory in Vienna (internally abbreviated VAB—not an acronym). Headed by Heinz Zemanek, the group originated in a research team at the Technical University of Vienna. Prior to the IBM move, the

---

[31] C. Strachey and M. V. Wilkes, "Some Proposals for Improving the Efficiency of ALGOL 60," *Communications of the ACM* 4, no. 11 (November, 1961): 488–91.

[32] Martin Campbell-Kelly, "Christopher Strachey, 1916-1975: A Biographical Note," *IEEE Annals of the History of Computing* 1, no. 7 (1985): 19–42.

[33] Christopher Strachey, "Curriculum Vitae", December 1971, Box 248, A.3, Christopher Strachey Collection, Bodleian Library, Oxford.

[34] T. B. Steel, ed., *Formal Language Description Languages for Computer Programming* (North-Holland, 1966). Other attendees included Adriaan van Wijngaarden, Edsger Dijkstra, Peter Naur, Saul Gorn, Brian Randell, and Doug McIlroy.

[35] Troy K. Astarte, "Formalising Meaning: A History of Programming Language Semantics" (PhD thesis, Newcastle University, 2019), Chapter 4.

[36] R. E. Utman, "Minutes of the 4th Meeting of IFIP WG 2.1", September 1964, Ershov Archive, http://ershov.iis.nsk.su/en/node/778103.

group had developed an ALGOL 60 compiler for their Mailüfterl computer. Two members of the group recorded their experience in a technical report; they felt that without a "complete and unambiguous" formal definition, the task was long and difficult.[37] Zemanek, chair of the FLDL organising committee, staffed the conference with members of the VAB, seeing an opportunity for his team to learn the state of the art in language formalism.[38] They needed this, because in late 1964 the VAB had agreed to write a formal definition of IBM's new programming language, PL/I. This was an ambitiously large and complex language intended to "meld and displace FORTRAN and COBOL," freeing IBM from the need to maintain compilers for multiple languages.[39] Vienna was instructed to develop a formal definition that would aid the compiler development.[40]

The first version, finished in 1966, formalised nearly every aspect of the syntax and semantics of PL/I.[41] It was a huge document—over 300 pages—known internally as the 'Vienna telephone directory'.[42] Drawing influence from McCarthy and Landin, PL/I was defined in terms of its actions on a large and complex abstract machine.[43] Successive versions refined the approach, adding in abstractions of storage and the new features constantly being added by language control.

---

[37] Peter Lucas and Hans Bekič, "Compilation of ALGOL, Part I—Organization of the Object Program," Laboratory report (IBM Laboratory Vienna, 1962).

[38] Peter Lucas, "Formal Semantics of Programming Languages: VDL," *IBM Journal of Research and Development* 25, no. 5 (1981): 549–61; Heinz Zemanek, interview by William Aspray, 1987.

[39] Fred Brooks, in Len Shustek, "An Interview with Fred Brooks," *Communications of the ACM* 58, no. 11 (November 2015): 40.

[40] R. A. Larner and J. E. Nicholls, "Plan for Development of Formal Definition of PL/I" (IBM internal memo, September 1965).

[41] ULD-III, "Formal Definition of PL/I (Universal Language Document No. 3)" (IBM Laboratory Vienna, December 1966).

[42] Saul Rosen, "Programming Systems and Languages 1965-1975," *Communications of the ACM* 15, no. 7 (1972): 592. To see examples of the style, see Astarte and Jones, "Formal Semantics of ALGOL 60".

[43] Lucas, "Formal Semantics of Programming Languages".

By 1968 the Vienna group had defined the entirety of a complicated language, managing its storage, tasking (concurrency), scoping, and rich type system. The VAB had produced a definition of ALGOL 60 and a notational guide,[44] and, critically, the PL/I definition had been used to prove properties about an implementation.[45] Later work by Henhapl used the definition to find bugs in the compiler; he and Jones, temporarily visiting the Lab from compiler development in Hursley, proposed an algorithm to fix them.[46] These were first steps towards a use for formal semantics: while the VAB had demonstrated "practical feasibility" for proofs about a compiler, they became "unsurmountably [sic] lengthy and tedious" when attempted on a full-size language like PL/I.[47] The Viennese work was not received positively and mostly served as a counter-example in many technical responses: a nightmarish behemoth to scare readers into preferring shorter formalisms.[48] Both Strachey and Hoare cited Viennese work but only in the context of an argument that a better way to define semantics must exist.[49]

Tony Hoare began his career in computing working for Elliot Brothers, where he wrote an ALGOL 60 compiler.[50] He had learnt about ALGOL at a workshop in Brighton taught by Landin, Dijkstra, and Naur. Hoare was impressed by the ALGOL report's formal syntax and

---

[44] Peter E. Lauer, "Formal Definition of ALGOL 60" (IBM Laboratory Vienna, December 1968); Peter Lucas, Peter E. Lauer, and H. Stigleitner, "Method and Notation for the Formal Definition of Programming Languages" (IBM Laboratory Vienna, June 1968).

[45] Peter Lucas, "Two Constructive Realisations of the Block Concept and Their Equivalence" (IBM Laboratory Vienna, June 1968).

[46] W. Henhapl and C. B. Jones, "Some Observations on the Implementation of Reference Mechanisms for Automatic Variables" (IBM Laboratory, Vienna, May 1970); W. Henhapl and C. B. Jones, "The Block Concept and Some Possible Implementations, with Proofs of Equivalence" (IBM Laboratory Vienna, April 1970).

[47] Kurt Walk, "Roots of Computing in Austria: Contributions of the IBM Vienna Laboratory and Changes of Paradigms and Priorities in Information Technology," in *Human Choice and Computers* (Springer-Verlag, 2002), 82.

[48] Astarte, "Formalising Meaning", Chapter 5.

[49] Robert Milne and Christopher Strachey, "A Theory of Programming Language Semantics" (Privately circulated, 1974);

[50] C. B. Jones and A. W. Roscoe, "Insight, Inspiration and Collaboration," in *Reflections on the Work of C.A.R. Hoare*, ed. Cliff B. Jones, A. W. Roscoe, and Kenneth Wood (Springer-Verlag, 2010), 1–32.

wanted to solve semantics similarly formally.[51] Hoare joined the ALGOL committee, and was also present at FLDL where he made a comment that presaged his later work. Hoare noted that any complete implementation of a language (such as a compiler) could be viewed as a precise definition, and vice-versa. However, for real utility, a definition should not be concerned with machine-specific details, and furthermore should have "a mechanism for failing to define things".[52] In other words, Hoare explicitly argued for the utility of abstraction over embodiment when considering a programming language.

Hoare began sketching an alternative to the Vienna approach while attending a 1965 workshop on the method. Citing inspiration from Peano's axioms of arithmetic, Hoare directly opposed the 'constructive,' state-based style used by Landin and the VAB, arguing instead for choosing appropriate basic concepts—axioms—that represented key properties of programming languages. Hoare's approach was presented in contrast to a state-based one because of the difficulty in specifying every single aspect of a state. Rather, Hoare advocated specifying only properties of interest for a given program construct and defining the feature by its effect on these properties.[53]

After working through a series of unpublished drafts for a semantic method,[54] Hoare first presented his work at a 1969 meeting of IFIP's Working Group 2.2, which had been formed after FLDL as a continuation of that conference's formal language description agenda. Hoare's 'axiomatic method' met with much interest, positive and negative.[55] Some praised the simplicity

---

[51] Hoare, interviewed in Edgar G Daylight, "From Mathematical Logic to Programming-Language Semantics: A Discussion with Tony Hoare," *Journal of Logic and Computation* 25, no. 4 (2013): 1091–1110.

[52] C. A. R. Hoare, in *Formal Language Description Languages for Computer Programming* (North-Holland, 1966), 142–43.

[53] Cliff Jones and Troy Astarte, "Challenges for semantic description: comparing responses from the main approaches", in *Proceedings of the Third School on Engineering Trustworthy Software Systems*, ed. Jonathan P. Bowen, Zili Zhang, and Zhiming Liu (Springer-Verlag, 2018).

[54] For example, C. A. R. Hoare, "The Axiomatic Method: Part I" (Unpublished, December 1967).

[55] Walk, "Minutes of the 3rd Meeting of IFIP WG 2.2 on Formal Language Description Languages."

and elegance of the approach, while others were sceptical about its ability to scale to more complex features.

At the same meeting, Christopher Strachey met Dana Scott. Strachey had continued working on language description after his consultancy closed and presented some ideas at FLDL.[56] Influenced by his experience with CPL and Landin's lambda calculus work, Strachey started by considering properties of machine storage, deciding that functions were an appropriate abstraction for the core of computation. Computer stores could be seen as functions mapping addresses to values, and statements were functions mapping stores to stores. Like McCarthy, Strachey used lambda calculus to represent these functions, and he called his method 'mathematical semantics.' Strachey had not worked out an appropriate basis for these functions, however, and he later acknowledged his approach was at that time, "deliberately informal, and, as subsequent events proved, gravely lacking in rigour".[57] Scott, a logician who had studied under Alonso Church, was intrigued by Strachey's propositions at Working Group 2.2, and joined his Programming Research Group (PRG) in Oxford for the autumn term of 1969.

Thus began a period described by Scott as "feverish activity" and "one of the best experiences in [his] professional life."[58] Scott developed a theory of domains with the properties required to model programming languages, and this gave a foundation to Strachey's semantics.[59] Further extensions to the approach were made by students and researchers at the PRG, and by 1974 it was used to model ALGOL 60[60] and the large pedagogical language Sal.[61] Scott/Strachey semantics was more concise than the Vienna work, but involved complex mathematics—and

---

[56] C. Strachey, "Towards a Formal Semantics," in *Formal Language Description Language for Computer Programming*, ed. T. B. Steel (North-Holland, 1966), 198–200.

[57] Quoted in Dana S. Scott, "Some Reflections on Strachey and His Work," *Higher-Order and Symbolic Computation* 13, no. 1 (2000): 110.

[58] Dana S Scott, "Logic and Programming Languages," *Communications of the ACM* 20, no. 9 (1977): 637.

[59] Dana Scott and Christopher Strachey, "Toward a Mathematical Semantics for Computer Languages," (Monograph, Oxford PRG; 1971).

[60] Peter David Mosses, "The Mathematical Semantics of ALGOL 60" (Monograph, Oxford PRG, January 1974).

[61] Milne and Strachey, "A Theory of Programming Language Semantics."

inclusion of features like jumps required extensive convolutions of the functions. Although others carried forward research in mathematical semantics, Strachey's contributions came an abrupt end in 1975 with his sudden death.

Community commitments

Strachey's decision to label his semantics 'mathematical' (despite his concession that he was "not a mathematician"),[62] exemplifies the desire amongst semanticists to provide a "mathematical theory of computation" to legitimise the emerging field of computing.[63] As McCarthy had referenced Newton and Kepler, so Hoare evoked mathematical 'greats' to justify his work: "[this approach] may represent the same magnitude of advance in Computer Science as the axiomatic geometry of Euclid compared with the crude land-measurement of the ancient Egyptians."[64] Zemanek, too, gave his group's work a prestigious Viennese intellectual lineage, referring to the *Wiener Kreis* and Wittgenstein.[65]

As computers became involved in the practice of writing[66] practices were borrowed further from linguistics. Formal linguists such as Chomsky were invited to the FLDL conference; VAB's Lucas later wrote that the view was that "by viewing computers as language interpreting machines it [became] quite apparent that the analysis of programming (and human) languages [was] bound to be a central theme of Computer Science."[67]

---

[62] Quoted in Roger Penrose, "Reminiscences of Christopher Strachey," *Higher-Order and Symbolic Computation* 13, no. 1 (2000): 83.

[63] Scott also used the phrase, in D. Scott, "Outline of a Mathematical Theory of Computation" (Monograph, Oxford PRG, November 1970).

[64] Hoare, "The Axiomatic Method," 1.

[65] See for example in Thomas B. Steel, "Preface," in *Formal Language Description Languages for Computer Programming* (North-Holland, 1966).

[66] Stephanie Dick, "Machines Who Write [Think Piece]," *IEEE Annals of the History of Computing* 35, no. 2 (2013): 88.

[67] Peter Lucas, "On the Formalization of Programming Languages: Early History and Main Approaches," in Dines Bjørner and Cliff Jones, *The Vienna Development Method: The Meta-Language*, vol. 61, Lecture Notes in Computer Science (Springer-Verlag, 1978), 3. See also David Nofre, Mark Priestley, and Gerard Alberts, "When Technology Became Language: The

Bringing tools and techniques from one field to another also carries the source field's agenda.[68] Tarski and Carnap's semantics of formal logic influenced formalisation in computing; the bridging of these practices and the formalist agenda generated what some saw as a new science of computing.[69] However, as Babintseva observes, mathematicians do not rely solely on demonstrative reasoning and indeed computing formalists show a greater obsession with formality than is present in mathematics.[70] DeMillo and Lipton, critics of formalism, explained the pervasiveness of this formalistic attitude: "the amount of influence that [formalism] had in computing at the time was enormous. It was *the* only way to understand programs; it was *the* only way to understand the foundations of programming".[71]

We might observe a kind of performative mathematisation in the works discussed here, where references to mathematics are used to legitimise the authors' own beliefs in the "theory of programming language semantics" or "axiomatic basis for computer programming". This is evidenced by Strachey's initial unconcern about the lack of foundation for his function-based approach, and McCarthy's incorrect application of lambda binding. Hoare, too, initially presented his system of axioms without consideration of consistency or completeness which made it gravely lacking by mathematical standards.

There was tension within the formalist community about precisely which values embodied their mathematical ideals, as the following exchange in response to Hoare's WG 2.2 presentation shows:[72]

Caracciolo: A reduction to simpler questions would mean to omit the proper problem.

Origins of the Linguistic Conception of Computer Programming, 1950–1960," *Technology and Culture* 55, no. 1 (January 2014): 40–75.

[68] Michael S Mahoney, "Computer Science: The Search for a Mathematical Theory," in *Histories of Computing*, ed. Thomas Haigh (Harvard University Press, 2011), 128–46.

[69] Priestley, *A Science of Operations*, 230.

[70] Babintseva, "From Pedagogical Computing to Knowledge-Engineering".

[71] quoted in Tedre, *The Science of Computing*, 67. Emphasis original.

[72] Walk, "Minutes of the 3rd Meeting of IFIP WG 2.2 on Formal Language Description Languages."

> Scott: Only the most primitive, non-problematic things have been dealt with using this approach.
> Laski: A language definition should specify as little as possible.

Hoare was asked whether his method had been applied (so far) only to simple languages:

> Hoare: Yes. But, of course, difficult things are difficult to describe.
> Strachey: What is "difficult" very much depends on the frame-work of thinking. For example, assignment is difficult in the λ-calculus approach, recursion is difficult in other systems. But both occur in programming languages and are simple to use.

In other words, while there was agreement on the importance of abstraction, the choice of abstraction was not universally agreed. The intellectual values held by computer scientists came into conflict: elegance and parsimony ("specify as little as possible") contrasted with expressiveness and power ("cover as much ground as possible"). A subtler value is the cachet of working on problems that were recognized as difficult. Strachey's later description of his earlier work as "deliberately informal" and "lacking rigour" shows him framing his current work as more substantial and worthy—even though at the time he had derided the excess of formality in other works.

These values of intellectualism and abstraction were also in tension with a desire to present formal semantics work as practically important: embodying the formalisms would demonstrate their worth. Influenced by the product-directed atmosphere of IBM, the Viennese had PL/I compilers as ultimate end goals and their work on proof was intended to improve the reliability of these. Hoare wanted his work to be useful for the creation of machine-independent standards for programming languages;[73] and Landin saw his methods as a way to improve language design.[74] Strachey, meanwhile, called practical work performed without knowledge of

---

[73] Hoare, "The Axiomatic Method".

[74] P. J. Landin, "The Next 700 Programming Languages," *Communications of the ACM* 9, no. 3 (1966): 157–66.

fundamental principles "unsound and clumsy" and theory work with no connection to practical programming "sterile."[75] This goal of dual relevance was typical of computer science in the 1960s and 70s: many computing academics were theoreticians hoping to build an embodied science of computing by establishing a theoretical, abstract base and working upwards. This "reflective closure" was intended as a rationalisation of existing practice through theory-forming.[76]

Description vs. design

From the first presentations, formal semantics faced heavy criticism, most commonly that descriptions were too large and complex. This is illustrated by an exchange at an ALGOL meeting:[77]

> TURSKI: In Grenoble we decided that the proposed description method is a milestone in the development of the language.
> RANDELL: A milestone or a millstone?
> General laughter follows.

IBM's Language Control reacted similarly poorly to the 1966 PL/I definition, one member commenting that using the document to handle PL/I was akin to reading *Principia Mathematica* to perform addition.[78]

---

[75] Quoted in C. A. R. Hoare, "A Hard Act to Follow," *Higher-Order and Symbolic Computation* 13, no. 1 (2000): 71–72.

[76] Graham White, "Hardware, Software, Humans: Truth, Fiction and Abstraction," *History and Philosophy of Logic* 36, no. 3 (2015): 278–301.

[77] W. M. Turski, "Minutes of the 8th Meeting of IFIP WG 2.1", May 1967, Ershov Archive, http://ershov.iis.nsk.su/en/node/778136.

[78] G. W. Bonsall et al., "ULD3 and Language Development" (IBM internal memo to J. L. Cox., February 1967).

Frequently, this negativity was in response to models of whole programming languages. Toy examples showed it was relatively easy to give semantics to trivial chunks of programs, but at scale, semantic descriptions became unwieldy. The distinction between complexity in definition method and language under definition was often overlooked: much of the bulk in the ALGOL 68 and PL/I definitions was due to richness of features in those languages.[79] Strachey and Milne addressed this: "A superficial glance will show that our essay is long and our notation elaborate. The basic reason for this unwelcome fact is that programming languages are themselves large and complex objects which introduce many subtle and rather unfamiliar concepts".[80] In other words: "difficult things are difficult to describe."

The embodiment of formalism in language design was rare (ALGOL 68 is a notable example, but the language was hamstrung by its arcane presentation). Jones suggested that a reason for this was that semantics workers had not created a literature that was useable by and useful to language designers.[81] Plotkin agreed, explaining that engaging with formal language description was unnecessarily difficult for most language designers.[82]

Work on formal semantics did not cease as a result of this criticism,[83] and the ANSI PL/I standard, while not fully formal, shows clear inspiration from the Vienna work.[84] But many of the semanticists discussed here were disheartened by the reception to their work. McCarthy's interest in semantics dwindled at the tail end of the 1960s, partly due to the cool response his

---

[79] Astarte, "Formalising Meaning", Sections 5.7 and 7.1.

[80] Milne and Strachey, "A Theory of Programming Language Semantics," 22.

[81] In Cliff B. Jones et al., "Reminiscences on Programming Languages and their Semantics" (Panel at Mathematical Foundations of Programming Language Semantics XX, May 2004).

[82] Gordon D. Plotkin, interview by Troy Astarte and Cliff Jones, April 2018.

[83] Peter Mosses lists advances in the area in "The Varieties of Programming Language Semantics and Their Uses," in *International Andrei Ershov Memorial Conference on Perspectives of System Informatics.*, ed. Zamulin A. V. Bjørner D. Broy M., vol. 2244, Lecture Notes in Computer Science (Springer-Verlag, 2001), 165–90; and Peter O'Hearn and Robert Tennent collect more recent work in *ALGOL-Like Languages* (Springer Science & Business Media, 2013).

[84] ANSI, "Programming Language PL/I" (American National Standard, 1976).

ideas had received, and he instead became interested in Manna's program proving techniques.[85] This was typical: rather than giving up on abstraction, many reinvented themselves as program verificationists. Later, McCarthy reframed his work on formal semantics as "ultimately aiming at the ability to prove that an executable program will work exactly as specified."[86] He added he had believed that a situation would emerge where "no-one would pay money for a computer program until it had been proved to meet its specifications".[87]

The Vienna group's interests changed similarly throughout the 1970s. Having experienced difficulty proving theorems using their 1960s approach,[88] the group were given the chance to try again in late 1972, when IBM wanted a PL/I compiler for their new 'Future Systems.'[89] (The need for a commercial embodiment of their formalism was a constant source of friction). Jones re-joined the Vienna Lab from Hursley, and the group adapted Strachey's method. Using this 'denotational' approach, the group produced a definition of PL/I which was shorter and more amenable to formal reasoning.[90] The project ended abruptly when IBM cancelled 'Future Systems,' but Jones and colleagues collected the work into an edited volume.[91] The collection includes some discussion of programming language definition, but is more concerned with 'correct by construction' program development. This took ideas from the semantics work but working with proofs at a program level was easier due to the smaller scale of the problem.[92]

---

[85] John McCarthy, "Letter to A. P. Ershov" December 1965, Ershov Archive, http://ershov.iis.nsk.su/en/node/777964; John McCarthy, "Letter to A. P. Ershov", March 1968, Archive, http://ershov.iis.nsk.su/en/node/779549.

[86] Tedre, *The Science of Computing*, 155.

[87] Quoted in Mahoney, "Software as Science—Science as Software".

[88] Astarte and Jones, "Formal Semantics of ALGOL 60," 98.

[89] C. B. Jones, "The Transition from VDL to VDM," *Journal of Universal Computer Science* 7, no. 8 (2001): 631–40.

[90] Hans Bekič et al., "A Formal Definition of a PL/I Subset" (IBM Laboratory Vienna, December 1974).

[91] Bjørner and Jones, eds., *The Vienna Development Method: The Meta-Language*.

[92] Walk, "Roots of Computing in Austria."

Hoare's early drafts of his axiomatic method showed an emphasis on semantics, but when the ideas saw publication in 1969 his focus had shifted. The paper highlighted a "logical basis for proofs of the properties of a program" with language definition relegated to the final section. Hoare became one of the loudest advocates of the 'strong verificationist' position: that all computing was reducible to mathematics and could be subject to formal proof.[93] Hoare's position was shared by Edsger Dijkstra, who viewed programming languages as mathematical objects with important aesthetic properties,[94] and had worked for a time on formal semantics.[95] Like Hoare, however, Dijkstra's agenda shifted from languages to programs; while his work on "predicate transformers" included mention of semantics, the aim was "the goal-directed activity of program composition."[96]

This change in 'technocratic paradigm' from language to program was typical of the 1970s.[97] An example is the formation of IFIP's Working Group 2.3, which followed the publication of ALGOL 68 as an IFIP-sponsored language. A significant minority of the ALGOL Committee were concerned the ALGOL 68 team had become too focused on their language and its definition method, losing sight of what the Minority Group felt to be the real goal: enabling good programming.[98] These members signed a 'minority report' (drafted by Dijkstra)[99] and formed a new Working Group, whose remit was 'Programming methodology.' WG 2.3 was something of an "elite member's club," with gate-kept entry and a deliberately relaxed

---

[93] Tedre, *The Science of Computing,* Ch. 4.

[94] Gerard Alberts and Edgar G. Daylight, "Universality Versus Locality: The Amsterdam Style of ALGOL Implementation," *IEEE Annals of the History of Computing* 36, no. 4 (2014): 52–63.

[95] E. W. Dijkstra, "On the Design of Machine Independent Programming Languages," Report (Mathematisch Centrum, October 1961).

[96] E. W. Dijkstra, "Guarded Commands, Nondeterminacy and Formal Derivation of Programs," *Communications of the ACM* 18 (1975): 457.

[97] Amnon H. Eden, "Three Paradigms of Computer Science," *Minds and Machines* 17, no. 2 (July 1, 2007): 135–67.

[98] Peláez Valdez, "A Gift from Pandora's Box."

[99] Dijkstra Edsger, "Minority Report", 1969, Ershov Archive, http://ershov.iis.nsk.su/en/node/805785.

atmosphere.[100] The initial chair, Mike Woodger, noted "programmers needed tools other than bigger and better programming languages";[101] and rather than focusing on languages, proving program correctness was one of the group's chief concerns.

Around the same time, IFIP WG 2.2 on language description went on hiatus following a period of identity crisis.[102] One émigré was Cliff Jones, who moved to WG 2.3 in line with his own changing priorities;[103] Hoare, Strachey, and others moved at the same time. Summaries of the first decade of WG 2.3 meetings show a number of similar talking points to WG 2.2, including formal semantics.[104] These were, however, discussed as general *concepts* rather than in the context of particular programming languages. Even as the focus among formalists changed from languages to programs, the emphasis on abstraction and mathematics had not gone away.[105]

Conclusions

The history of formal semantics is a key part of the history of programming languages and program verification. Formal semantics offered a way to understand programming languages and programs that had been decoupled from machines and stood as a chief component in a theory computation. Crucially, this is a European narrative.[106] In the USA, complexity theory dominated the academic discourse and McCarthy's work on semantics did not impress his American colleagues.[107] Formal semantics works fits instead within the ALGOL research programme, a

---

[100] Haigh, "Assembling a Prehistory for Formal Methods."

[101] M. Woodger, "A History of IFIP WG 2.3: Programming Methodology," in *Programming Methodology: A Collection of Articles by Members of IFIP WG 2.3*, ed. David Gries (Berlin, Heidelberg: Springer-Verlag, 1978).

[102] Astarte, "Formalising Meaning", Section 7.2.

[103] Cliff Jones, personal communication, 2020.

[104] Woodger, "A History of IFIP WG 2.3: Programming Methodology."

[105] Peláez Valdez, "A Gift from Pandora's Box," 198.

[106] Astarte, "The History of Programming Language Semantics", Section 8.

[107] Dick, "Computer Science"; Mahoney, "Computers and Mathematics."

peculiarly European initiative.[108] This was noticed at the time: McIlroy recalls a stark difference between the people he met at FLDL and his American colleagues: "it was real computer science, in Europe. There were no real computer scientists in the US at the time." [109] The national computing cultures in Europe, with their focus on formal logics and mathematics, shaped a different interpretation of the science of computing, challenging the US-dominated narrative.[110]

Semantics work was also advanced as a tool with practical implications: compiler creation and language design. However, the complexity of embodying the abstractions of formal semantics in full-scale programming languages prevented immediate use. Instead, as Hoare outlined, there was a slow bridging of practice from formal semantics and verification: over-engineering, bug-checking, and the associated mindset of structured programming.[111]

The 1970s saw a change amongst semanticists, who came to realise that programming languages were "difficult things" which were "difficult to describe." Instead, a new emphasis came on applying formalism at the program level, a smaller-scale challenge: in a kind of academic sleight of hand, specifying languages was substituted for specifying programs. Although the focus changed, the researchers involved continued to claim the same end results: mathematised computing and more reliable programs.

The relationship between formal semantics and verification had in a sense switched. While the ability to write proofs was once a goal of a language description, now the role of formal semantics had become instead a part of verification: the fundamental basis for some formal proofs.[112] The intellectual (or even moral) commitment to provability as a form of knowability shows a continuity in the ontology of programming shaped initially by formal

---

[108] David Nofre, "Unraveling Algol: US, Europe, and the Creation of a Programming Language," *IEEE Annals of the History of Computing* 32, no. 2 (2010): 58–68.

[109] Doug McIlroy, interview by Cliff Jones, May 2018.

[110] Babintseva describes an alternative Russian view of computified thought in "From Pedagogical Computing to Knowledge-Engineering", this volume.

[111] C. A. R. Hoare, "How Did Software Get So Reliable Without Proof?" in *International Symposium of Formal Methods Europe* (Springer-Verlag, 1996), 1–17. That this process took time, Hoare argued, should not have been surprising in retrospect, since deploying brand new research methods in commercial products carried significant risk.

[112] Dana Scott, interview by Tony Dale, April 1994.

semantics work, and the fervour of narratives like 'Grand Challenges in Verification' persists well into the 21st Century.[113]

Acknowledgements

Many thanks to Janet Abbate and Stephanie Dick for helpful editorial remarks; Cliff Jones and David Dunning for feedback on drafts; and Margaret Gray for advice and copy editing. The paper emerged from a talk at the 5th *History and Philosophy of Computing* conference; thanks to the attendees who made comments. This research was supported by an EPSRC studentship and writing this paper by a Leverhulme Trust grant RPG-2019-020.

[113] Tony Hoare and Robin Milner, "Grand Challenges for Computing Research," *The Computer Journal* 48, no. 1 (January 2005): 49–52.