Prepublication proofs:
Martin Irvine, "Semiotics in Computation and Information Systems." In *Bloomsbury Semiotics*, Jamin Pelkey, ed., 4 vols; vol 2:203–37. London: Bloomsbury Academic, 2022.

CHAPTER NINE

Semiotics in computation and information systems

MARTIN IRVINE

INTRODUCTION: DEBLACKBOXING COMPUTER SYSTEMS AS SEMIOTIC SYSTEMS

Computing and semiotics have been inextricably connected since the late seventeenth century. The intellectual history of *computation* is not a story about *machines*, but about discoveries in the structures of symbolic thought, specifically how the patterns of necessary reasoning in logic and mathematics can be *formally symbolized* at different levels of abstraction, and then *physically* 'operationalized' by assigning *symbolic* structures to *physical* structures. As leading historians of computing explain,

> The modern computer was not the inevitable outcome of technological advance. The crucial prerequisite for the useful application of technology to computing was the development of notation, or language systems, sufficiently comprehensive to satisfy both the need for representation, and the need to express and implement mechanisms for the transformation of expressions in the language. […] The real intellectual origin of the modern computer has much deeper roots in the themes of representation and of automatic methods of symbolic transformation.
>
> (Campbell-Kelly and Russ 1994: 701, 703)

These special 'language systems' for 'automatic methods of symbolic transformation' (what we know as *computer code + data*) extend back to what Leibniz called a 'mechanical thread' (thinking with symbols that represent necessary patterns in logic and mathematics). We can trace this 'thread' from the era of Leibniz's philosophy of symbols, his model for an arithmetical calculator, and a method for calculating with the binary (base 2) number system, including his design for the first binary calculator (*c.*1700),[1] through the era of Charles Babbage, George Boole and C. S. Peirce (1830s–1910s) (origins of formal logic and mechanical calculating 'engines') (Gabbay and Woods 2004), and on to the era of modern mathematical logic, the foundations of the modern electronic computing era, and digital information (1930s–50s) (Gabbay et al. 2014). Leibniz's 'thread' appears in all physical devices designed to *implement* symbolic processes by *assigning* and *delegating*

their representations and operations (mapped out in special symbols) to intentionally designed, corresponding components (Hilton 1963; Davis 2012; von Plato 2017).

The words *compute* and *calculate* were synonymous until recently. Both are derived from Latin words that refer to methods for counting with number symbols and doing arithmetic, and the term *computer* originally meant a *person* who did calculations with numbers and other formal symbols (Grier 2005). In fact, all designs for physical 'computer systems' (both earlier and modern) are extrapolations from how *human computers* in the nineteenth and early twentieth centuries worked out calculations with numbers, notation systems, formulas, calculating devices and reference books with 'look-up' tables of logarithms, trigonometry formulas and other pre-calculated values, to *perform* 'computations'. Human computers are the models for the first CPUs (Central Processing Units) in digital computers: a coordinating agency for interpreting data representations as 'inputs', applying step-by-step, rule-governed operations (logic 'outsourced' to logic circuits), then 'outputting' (writing out) results in further sets of symbols, and repeating the process as needed. C. S. Peirce was an expert in these methods, and both his scientific work as a 'computer' and his theoretical work in mathematics and logic became the foundation for his *semeiotic*.

The modern digital electronic computing era has all this history 'built in', and our 'computers' became technically possible with the unanticipated convergence of research and development in mathematics, logic, telecommunications and electrical engineering in the 1930s–40s (Ceruzzi 1983, 2003; Campbell-Kelly and Aspray 2014). This convergence was directly motivated by a core *semiotic* problem, one that weaves Leibniz's 'thread' through many layers of complex design solutions: granted that we want to use electrical signals and components for speed and scalability in computation, how can we structure electricity and physical components to *represent tokens of symbols* to be computed, and then assign *operations* (necessary *interpretations*) that 'go with' the symbol tokens, and direct the system to *transform* the tokens as first represented ('inputs') into new tokens that represent the values or meanings as 'computed results' ('outputs'), in a controlled, automatic process? The design solution for this semiotic problem *is* the story of modern computing, right down to all the devices, networks and media we use today. As Licklider explained:

> Digital computers deal essentially with discrete patterns that may represent names or pictures quite as readily as numbers, and […] numerical calculation is merely one of many things that processors of discrete patterns can do. For our purposes, it is beside the point that the main early applications of digital computers were numerical. It is more significant that textbooks now call them 'general symbol processors'.
>
> (Licklider 1968: 274)

How we get from the earlier room-size 'number crunching' electronic computers of the 1950s–60s to our contemporary computer systems for 'general symbol processing' is a story of *applied semiotics*:

> [T]he domain of computation actually comprises symbols – by which I mean things that represent other things (for example, a string of alphabetic characters) […] The act of computation is, then, symbol processing: the manipulation and transformation of symbols. Numbers are just one kind of symbol; calculating is just one kind of symbol processing. And so, the focus of automatic computation, Babbage's original dream, is

> whether or how this human mental activity of symbol processing can be performed by (outsourced to) machines with minimal human intervention. Computer science as the science of automatic computation is also the science of automatic symbol processing.
>
> (Dasgupta 2014: 12)

We will unpack the assumptions about 'symbol processing' here, and complete this brief description to explain how computing now includes all digitally representable human symbol systems by filling in the details with C. S. Peirce's *semeiotic*.

Since the 1960s, and more extensively since the 1990s, modern semiotics and computing theory intersect across a wide interdisciplinary field that includes research programs in computer science, systems theory and engineering, design theory, philosophy, cognitive science, linguistics, logic and mathematics, as well as interdisciplinary work in the field of semiotic studies.[2] In this context, an important viewpoint is emerging: everything in computing and information systems is based on underlying design principles for semiotic systems, which include structures for interactions and communications between and among semiotic agents (*human* and *delegated agents* in software). This chapter provides an orientation to this exciting and expanding field of study.

The best way to make 'computing and semiotics' accessible for students and non-specialists is through a unifying framework that reveals how semiotic functions are correlated with the design principles for computing systems, digital media, programming and software, and user interfaces. But because all the relevant disciplines are constituted by multiple schools of thought with varying terminology, we need a framework with a generally consistent vocabulary for making useful syntheses of concepts and semiotic principles regardless of the specialized terminology of any subdiscipline. A unifying semiotic framework can be provided by combining three interrelated views of computing that enable us to focus on universal *design principles* for computer systems as *semiotic systems*:

1. A Peircean semiotic systems view: extending and applying the key concepts in Peirce's program of 'Logic as Semeiotic' (*c*.1902–12) for the computing systems that his work anticipated.
2. The systems and design view: combining Peirce's program for semeiotic with modern systems and design theory, as understood in all computing and information fields.
3. The cognitive-semiotic artefact view: defining the implemented designs in actual computer systems not as non-human 'machines' but as designed cognitive-semiotic artefacts, a view developed in cognitive science, anthropology and HCI (Human-Computer Interface Design).

This combined framework provides a 'deblackboxing' method for discovering *why* and *how* computing systems are intentionally *designed semiotic systems*, even though semiotic principles are hidden from view ('blackboxed') in the *implemented designs* of computer systems as *products*. The conceptual metaphor 'black box' was originally an engineering term for any component designed to take in certain kinds of inputs (energy, signals, information, etc.) and convert them into specified outputs (e.g., a radio, a voltage transformer, a codec for converting digital into analog audio/video): the *details* inside the components can just stay 'hidden' ('black-boxed', 'don't need to know', 'built-in'), because only the outputs matter for the design purpose. The concept is now universally

used in systems and software design: 'blackboxing' is used to *hide* the internal complexities of a module (at one functional level) that other modules in a system 'don't need to know about' for using the outputs communicated to system (see below on *systems and design theory*). But in our contemporary political economy for intellectual-property-protected products, this design principle is also used *intentionally* to close off access to computing systems in 'black-boxed' manufactured devices, which are intended to maintain 'users' as passive *consumers* blocked from understanding the *universal* semiotic principles on which the devices depend. A semiotic systems *de-blackboxing* method, then, is required for exposing the implemented design principles that are everywhere presupposed and actively instantiated, but, by historical accident, have been artificially closed off from users' understanding.[3]

The semiotic deblackboxing method introduced here exposes that all our interactions with computer systems are possible only by means of intentional logical 'mappings' between the levels of symbolic structures (with their interpretation processes) and corresponding levels in the design of computer systems (see below on *homology*). Briefly, the principle of *mapping* (correspondence relations between systems, domains or contexts) is used at many conceptual levels in mathematics and logic (e.g., functions, sets, diagrams, category theory), computer system design, software design and data design (databases, metadata schemes). Further, intra-system mappings are implemented in the physical structures of our devices (e.g., pixel coordinates mapped to graphics memory locations). For Peirce, *mapping* is a form of diagrammatic thinking in which relations among different levels of abstraction can be iconically represented, and also materially instantiated in designed artefacts (e.g., in actual maps and instruments).[4]

The semiotic foundations of computing systems can be described with both technical and conceptual accuracy, regardless of how computers and everything digital may be described in merely instrumental and operational language. I will therefore always use the term 'computer system', rather than 'computer', to remind us that we are always talking about *designed semiotic systems*, and not reified objects or products.

The semiotic systems view also allows us to make *implicit* semiotic assumptions *explicit*, like mapping out the unconsciously operational grammar of any language. The intellectual history of mathematics, logic and computer system design is interwoven with *implicit* and *tacitly presupposed* theories of signs, symbols and symbolic processes, and the framework outlined here allows us to recover these *implicit* semiotic principles and make them *explicit* for our understanding today.[5] As Peirce emphasized in many papers on disclosing the structures of logical inference in algebras and graphs, 'it is the chief task of logic gradually to develop that which is implicit in thought and step by step to make it explicit, or, at least, to show how to do so' (1897: MS 738.1). Peirce's whole program of *semeiotic* is an application of this logical method, and we will follow a Peircean 'step by step' method that allows to make the implicit *explicit* by turning things *inside-out*, exposing *how* and *why* computer systems are *designed semiotic systems*, whether or not they are expressly recognized as such.

By using this method, readers will notice that many authors who describe computer systems, user interface design, digital media and interactive software assume an underlying 'symbolic systems' view without presenting an explicit semiotic *theory*.[6] Important work based on semiotic principles continues at the intersections of cognitive science, philosophy, theoretical computer science and AI,[7] and all of this research and theory can be embraced and clarified in a unified Peircean semiotic systems view.

FRAMEWORKS FOR A SEMIOTIC SYSTEMS METHODOLOGY

Peirce's Logic as Semeiotic and semiotic systems

Other chapters will provide an overview of Peirce's semiotic theory (cross-references), but for our context, I will focus on key concepts in Peirce's works for describing computer and information systems as semiotic systems.

Our reference model for semiotic theory and computation will be Peirce's last version of his research program, termed 'Logic [Considered] as Semeiotic', which he did not live to complete (1890s–1913, and intensively during 1904–12).[8] Peirce's *semeiotic*, which must not be confused with post-1960s *semiotics*, was developed in the context of his work in mathematics, logic and scientific research (documented in Eisele 1979, 1985, and papers in NEM).[9]

Peirce can be considered the first 'computer scientist' in that most of his papers during the Logic as Semeiotic period include extensive drafts of his formal symbolic systems for representing necessary reasoning, and analyses of the possibilities for automated reasoning and 'logical (or reasoning) machines'. He envisioned *semeiotic* as a logical unification program for understanding the structure of all sign systems and symbolic reasoning, which was extensible to the logical-symbolic design principles of algebraic notation, graphs and diagrams, technical instruments and artefacts, logic machines, and all devices used for logical analysis and computation.

Peirce also had first-hand knowledge of the technologies and calculating machines of his era, and he designed his own scientific instruments that provided data for his computations. His contributions to Boolean logic and methods for formal symbolic notation became part of the symbolic logic tradition in the 1910s–30s,[10] which in turn became the foundation for the formal symbolic 'code' used in the first programming languages, which is now 'baked in' to the code libraries used in all contemporary programming languages. Peirce's grounding in mathematics, logic, instrument design and the logic machines of his era make his *semeiotic* the best extensible model for understanding the semiotic principles in the design of the computing and media systems that we use every day.

The systems and design view

The key concepts in Peirce's semeiotic can be readily combined with modern systems and design theory for developing consistent descriptions of computer and information systems as *designed semiotic systems*. This view requires a basic understanding of why and how digital computer systems are designed the way they are, rather than some other way.[11] System design theory includes the method of *levels of abstraction* for composing and decomposing system functions in multiple, hierarchical, interconnected *subsystems*, each designed to implement functions at different *levels* or *layers*, all of which *subserve* the purposes of an overall *architecture* (the master design) of a larger complex system.[12]

The method of *functional abstraction* is essential because it allows us to design a complex multifunctional system not as a totalized whole, but by distributing *functional levels* to corresponding modular *subsystems* (like processor and memory functions, and hardware/software modules for graphics and audio/video). Each subsystem is designed to perform a function and communicate with other subsystems through *interfaces* (transfer gateways) in the architecture. Behind what we perceive at the user-facing levels, computer system design is a way of 'orchestrating' multiple unobservable levels of *representation*

within the system, all supporting and returning to what we do observe, interpret and interact with.

An important pragmatist semiotic principle defined by Peirce underlies the unifying *architecture* of computer systems: the multiple levels of *subsystems* (also termed *modules*), from the most basic logic and memory components to software for representing interfaces for digital media, are designed to be 'orchestrated' as a *telic* (goal-directed, intentional, purposive) *system*. A computer system, by definition, *must* be an implementation of the purposes of semiotic agents (designers and users) directed into the whole system architecture. A computer system is made to exist only in service of the symbolic systems it is designed to instantiate, and the system is given telic direction in the way that *metasymbolic* programming code is designed to be interpreted as goal-directed *operations* in transition processes throughout the levels of the system (Gorn 1968, 1983; Horst 1996).

Recognizing how and why functional levels of abstraction are universally used in computer system architecture, digital information and software enables us to establish *semiotic levels of description* that directly correspond with *design levels* of computer systems. The designed systems view thus provides a key to making all the implicit and embodied semiotic principles explicit and systematically interpretable.

The cognitive-semiotic artefact view

Our framework reveals that 'computers' are not usefully defined as 'machines' at all. Combining concepts from cognitive science, philosophy of computation and anthropology, we find that computer systems are best defined as *designs* for *cognitive-semiotic artefacts*. An *artefact*, by definition, includes and presupposes its designers and makers. By making the principles for digital systems architecture accessible, we can reveal that a computer *system* is designed and implemented *by* and *for* semiotic agents.[13]

The cognitive artefact view also allows us to reveal how computer systems, information and networks exemplify what is now termed, in various fields, *extended, distributed*, *delegated* or *off-loaded* human cognition and agency.[14] Because design principles for everything computational are *telic* (purposive), computer system designs exist only for *implementing* delegated processes of symbolic cognition, which must always include shared physical-perceptible representations and a provision for ongoing dialogic interpretation (which we realize in physical input/output devices, interactive interfaces and communication across networks). This view also allows us to recognize the deeper history of contemporary information systems as part of the longer continuum of technologies designed for representing, storing and transmitting symbolic systems in physical media.

This framework allows us to reveal how computer *systems*, in designed levels of subsystems, are *structurally and constitutively semiotic*, and thus *different in kind* from anything else we call machines designed to perform other kinds of functions. Table 9.1 provides a system map for understanding computer systems levels and their corresponding implementations of semiotic levels.

Key concepts in Peirce's semeiotic: Sign systems, symbols, technical implementations

Important theoretical developments in Peirce's papers during his Logic as Semeiotic period apply directly to the design principles of modern computing systems, and it only a historical accident that the trajectory of thought developed in Peirce's writings was interrupted and then rediscovered, in part, in the 1930s–50s.[15] Peirce's important work

TABLE 9.1 Computer system levels: Overview of the semiotic system stack.

Computer systems architecture: Subsystems, component modules	*System functional levels: Levels of abstraction*	*Semiotic levels of description*
The top 'user-facing' level of multimedia interfaces mapped to input/output devices (screens, audio outputs) in continuous refresh cycles for projecting computational results and states of interaction.	Decoding and transducing digital data into analog (human perceptible) substrates (screens and audio outputs), as projections from software processes and symbol system tokenization mapping design (e.g., pixel patterns).	Semiotic agents interpret physical tokens of symbolic types in interface media, and direct further interpretations and representations; dialogic interactive interpretation and ongoing symbolic representations are physically instantiated with computer systems as 'co-agents'.
GUI software/hardware mappings for interaction, conducting inputs/outputs to/from system levels, and for dynamic updating of display and audio outputs.	Directing inputs (symbol token representations + intentions) and outputs (interpreted representations) in ongoing recursive process.	Semiotic agents direct the input/output structures in the physical subsystems for enacting dialogic interpretations, which project up to next level.
Active software and data levels. System modules orchestrate a 'running' software program by combining the levels of bytes in program code and bytes for data encoding (as indexed in different segments of active memory), for active operations on typed tokens.	Encoding symbol structures for *data* and *operations* ('symbols that mean' + 'symbols that do') in binary 'machine interpretable' byte representations in which operations transform tokens into new tokens in ongoing directed processes.	'Source code' program files (representing intentions in the metasymbolic code of a programming language) are translated by *interpretant* programs (compiler/interpreter) into binary code for active processes to be directed by semiotic agents.
Binary information encoded and interpreted as both *operations* mappable to processors and as *data types* (data encoding for all forms of digital media) accessible in memory.	Binary representations differentiated and defined for system functions. Data types as interpretations of the contents of memory locations.	Digital (binary) information as semiotic subsystem for tokenization and typed representations. Tokenization and retokenization of digital data from/to storage and active system memory.
Physical system architecture modules: processors, memory units, storage devices, internal I/O ('input/output'), interfaces to upper level modules. The level of minimal binary information structures.	CPUs and GPUs for logic and programming instructions, and digital memory devices for holding long-term representations (storage), and short-term (active RAM) data representations in physical locations indexed in the system.	Physical tokenization of symbolic structures for both data and operations. Every component at the first level implements a mapping of symbol structures and operations, the outputs of which are communicated 'up' the system stack.

on these topics has not yet been recovered for current research and theory. Some of the most important concepts are summarized here.[16]

Peirce frequently stated that his unified theory would embrace all forms of thought, reasoning and logic based on systems of 'external signs', including technologies designed to perform symbolic actions and relations (e.g., 1909: MS 637). He emphasized the constitutive physical-perceptible and cognitive-logical structures of all sign systems, and the necessity of a semiotic agency which activates triadic relations in ongoing generative interpretants. '[E]very reasoning is of the nature of a sign […] *Sign* will here be the general name for everything [used in reasoning], whether it be an instrument of music, a mental resolve, a voyage of discovery, or anything else that plays an essential part in the spread of intelligence' (1907: MS 602.7–8; cf. 1904: MS 774, EP 2.326). Peirce continually refers to technologies that are based on semiotic and logical principles; for example:

> Reason […] only acts through signs, spoken or written or 'scribed' or imagined. That which has made all our wonderful engines, wireless telegraphs, telephones, phonographs, and a thousand other wonders possible, has been the differential calculus, by which scientific men are instructed how to make the experiments that will be important. What is this 'differential calculus'? It is a system of signs invented by the great philosopher Leibniz.
>
> (*c*.1911: MS 514.46–7)

Peirce had thorough knowledge of the electrical signals technology of his era: he designed measuring instruments that used electromagnetic switches (as in telegraph systems), he drew the first diagram for using electrical switches to perform Boolean logic operations (1886: W5.421–3, and see Gardner 1958; Ketner and Stewart 1984), and he developed a binary system for encoding and encrypting Morse code (*c*.1902: MS 1361). Peirce clearly understood how binary logic maps onto switched electrical circuits, and how a binary mathematical code could be used for electrical signals, but these applied semiotic ideas had to wait for their application in the 1930s, when rediscovered by Claude Shannon for telecommunication networks and binary data (Shannon 1938; 1948).

Peirce also described how telecommunications signals create a *semiotic subsystem* that combines physical sign tokens, physical interpretants and human interpreters:

> Every thought, or cognitive representation, is of the nature of a sign. 'Representation' and 'sign' are synonyms. The whole purpose of a sign is that it shall be interpreted in another sign; and its whole purport lies in the special character which it imparts to that interpretation. When a sign determines an interpretation of itself in another sign, it produces an effect external to itself, a physical effect, though the sign producing the effect may itself be not an existent object but merely a type. […] Some signs are interpreted or reproduced by a physical force or something analogous to such a force, simply by causing an event; as sounds spoken into a telephone effect variations or the rate of alternation of an electric current along the wire, as a first interpretation, and these variations again produce new sound-vibrations by reinterpretation [… T]he rate of alternation of an alternating current along the wire [is] a series of variations making up a sign that interprets, i.e. translates, the acoustic sign, and in its turn setting up new acoustic vibrations in the receiver, as a reinterpretation.
>
> (1904: MS 1476.4–5)

This statement is not only a summary of Peirce's semiotic theory, but a clear example of how his *semeiotic* includes any technical design 'that plays an essential part in the spread of intelligence'.

We continue to use electrical signals for tokenizing symbolic structures in digital electronics, for which digital-analog decoders and transducers 'translate' the binary symbolic sequences to and from human perceptible forms (all audio and visual media). In fact, Peirce is describing the elements of the electrical signals system that became the foundation of modern telecommunications and information theory, and the electronic representations required for digital computing that soon followed. Solving the problem for physically retokening (reproducing) electrical signals in predictable structures is what gave us Claude Shannon's information theory (measuring and quantizing information in digital *bits*), and, ultimately, gave us the internet packet design principles that solved the problem of retokening digital data units across unlimited network connections.

In his many discussions of the potential for automating necessary reasoning in formal symbol systems, Peirce continually emphasizes that such a system may be possible if it can be 'self-controlled' like human cognitive control over a logical process. Peirce explains that a *symbol*, as an intelligible representation given a physical existence in time,

> may, in its capacity as such, produce effects in the material universe [… It] can have a history, may be affected by associations with other signs, and gradually may undergo a great change of meaning, while preserving a certain self-identity. Indeed […] connected with suitable machinery or other physical organism, being able to produce external effects by virtue of its signification, [a symbol] may by one branch of its signification act upon another branch of its signification; and there we have the first step toward self-control.
>
> (1905: MS 290.60)

This description is close to our concepts for data representations and the metasymbolic levels of code interpreted in a computer program. Further, since we can consider 'that a man is a machine with automatic controls' it could be possible to delegate rational self-control in a design for a mechanical symbolic process: 'This operation of self-control is a process in which logical sequence is converted into mechanical sequences […]. There is a class of signs in which the logical sequence is at the same time a mechanical sequence' (1905: MS L 390.39).

These references are only a small sample from Peirce's extensive writings on Logic as Semeiotic that reveal how the ideas for modern computing systems are based on a rediscovery of Peircean principles. In fact, as if answering Peirce's requirement that an automated logical system must be a 'self-controlled', i.e., regulated as a telic (purposively directed) system, the design principles for modern computer architecture in the 1940s–50s solved the problem of *internal* logical control (CPUs), and interactive programming design since the 1970s solved the problem of semiotic agents c*ontrolling and directing* running software as an active, ongoing, dialogic process. Peirce's inclusive model of Logic as Semeiotic provides the semiotic structural details for describing modern digital computer systems as designed semiotic systems.

Table 9.2 presents a brief summary of important terms and concepts in Peirce's writings on semeiotic in 1902–12 that apply directly to concepts used in modern computing systems.[17]

TABLE 9.2 Important concepts in Peirce's writings on *Logic as Semeiotic* (1902–12).

Triadic structure of signs/symbols in systems of relations	Peirce extends his concepts for triadic symbol structures for theory on the properties of physical instantiation and sign actions, kinds of interpretants, the interpretant function as a generative principle, and semiotic agency.
Type/Token Relation	From 1906 on, Peirce used the terms *token* (physical instance) and *type* (general abstract pattern) for the correlation of perceptible representations to intelligible symbolic forms (replacing his earlier terms). Anything symbolic must be tokenizable and retokenizable as shared cognitive anchors.
Physical properties of signs, and how signs can be used to cause physical actions	Peirce describes how all signs/symbols require instantiation as physical-temporal-perceptible and intersubjectively available representations. Peirce's concepts (representamen, tokens, physical indices) can be generalized as the principle for physical *structured substrates*, which hold perceptible patterns. Symbols thus instantiated can be used to cause physical actions in a designed system.
The dialogic principle	Peirce extends the model of meaning-development in interpretants in human dialog to the steps performed in logic as interactive interpretation with diagrams, notation systems, and technical devices. Externalized structures (graphs, diagrams, notation systems for computation) that support dynamic interpretation for reasoning are termed *Quasi-minds*.
Symbols and metasymbols: Formalizing necessary reasoning and potential for automation	From the 1880s on, Peirce developed concepts for formal symbols and symbols used at different levels of abstraction, which (in modern terms) are *metasymbols* for logical operations, abstractions, syntax and inference rules, in both algebraic and diagrammatic (graph) notations. From 1902 on, he describes how assigning the rule-governed operations of formal symbols to physical structures could enable automation in a self-controlled system.
Boolean logic and binary (base 2) number system	Peirce drew the first diagram for performing Boolean AND OR operations with electromagnetic switches (1886), and in the 1880s–90s he became the leading American authority on Boolean symbolic logic. From 1904–12, he wrote hundreds of pages of mostly unpublished work on Boolean operations, the base 2 number system, and methods for binary computations.

HISTORIES OF DISCIPLINES

Major contributions to semiotics and computing, Leibniz to multimedia

There is a long intellectual history of semiotic thought before 'semiotics' as a post-1960s academic field (Eschbach and Trabant 1983; vols. 1–2 of Posner et al. 1997–2003), and this history is interwoven with a parallel history of computation (theory, design and implementation) before and after contemporary digital computers. Tracing the semiotic archaeology of computing with the framework presented here is important for our deblackboxing approach.

Table 9.3 presents a conceptual overview of major developments in this combined history of theory and technical implementations so that the underlying *semiotic systems* design principles can become accessible for further study.[18]

TABLE 9.3 Major developments in computing and semiotics: Theory and implementations, Leibniz to the Internet.

1679–1710	G. W. Leibniz, philosophy of symbols and symbolic operations; designed calculating machine, and model for a binary (base 2) calculator.[19] Leibniz was a major influence on C. S. Peirce.
1820–50	Charles Babbage develops mechanical Difference Engine and designs for the Analytical Engine (1820s–40s). Develops symbolic notation system for symbolic-mechanical homologies.
	1832–37: Samuel Morse designs Telegraphic Code as proto-binary 'system of signs' mapped to open/closed circuits in electromagnetic switches. The foundations for all future electronic communications.
	1847–54: George Boole, *Laws of Thought*: the binary (two-value) algebra of logic.
1860s–1910s	C. S. Peirce:
	Develops and expands Boolean logic in algebraic and graphical systems for formal logic; develops a diagrammatic system of logic in 'Existential Graphs' (1880s–1912).
	Designs Boolean logic switches for a 'logical machine', and publishes an article on the design of 'Logic Machines' (1886–7: W 5.421–6, W 6.65–74).
	Develops methods for binary (base 2) computation (1904–12).
	Writes many drafts of papers for his unfinished program of 'Logic as Semeiotic' (1890s–1912), which includes semiotic concepts for technical systems.
1920s–30s	Beginnings of 'Information Theory' in telecommunications and electrical engineering: techniques for 'shaping signals', controlling electrical current and radio waves as a semiotic subsystem for transmitting representations.
1930s–40s	Alan Turing develops formal method for converting the steps in paper and pencil 'computations' into a discrete sequenced, automatable metasymbolic rule-governed process (1937).
	Claude Shannon rediscovers how to apply Boolean logic to electrical switches (1938); a design previously developed by Peirce. This design is developed further in the 1950s for clusters of 'logic gates' in all computing processors (CPUs).
	Charles Morris reinterprets Peirce's semiotics and pragmatism; his writings become known in engineering and science communities (Morris 1938, 1946, 1964)

1940s	Electrical engineering information theory and computer design theory converge.
	Design principles for electronic computing system architecture first developed.
	1945–48: John Von Neumann's architecture for processor + memory units established (Von Neumann 1987).
	Claude Shannon establishes the mathematical theory for discretizing information in electronics based on the binary *bit* as a substrate for information representations (1948).
	Vannevar Bush, *Memex* ('memory expander'): a design concept for bringing multiple symbol systems from different media into a unified, user-configurable desk 'display' (1945); a conceptual model for Engelbart and subsequent digital interface designs.
1950s–60s	John von Neumann, Arthur Burks, and colleagues standardize computing architecture for automating logic, and applying the binary system for data and code.
	Transition in computer science and engineering for reconceiving computer systems as general symbol processors for encoding any symbolic system, not merely 'number crunchers' (Hamming, Licklider, Engelbart, Newell and Simon).
	Donald MacKay, papers on Information Theory and Symbolic Systems (1952–68) (MacKay 1969).
	Colin Cherry, *On Human Communication* (1957), includes semiotic principles.
	Allen Newell & Herbert Simon develop 'physical symbol systems' theory: (1961, 1972, 1976, 2003); and Newell (1980, 1986); Simon (1993, 1996).
1960s–70s	J. C. R. Licklider, Doug Engelbart and Alan Kay redefine computing for multi-symbolic systems by developing engineering solutions for symbolic and interactive system concepts (see: Rheingold 2000; Moggridge 2007).
	Licklider redefines computers as a *metamedium* for symbolic systems, interactions, communication, and knowledge; funds Doug Engelbart's lab (1960–77).[20]
	Ivan Sutherland, *Sketchpad* (1963): proof of concept for graphical interface systems with a screen input device; display screen reconceived as two-way interface.
	Doug Engelbart, 'Augmenting Human Intellect' research program at SRI: computer systems redesigned as multi-symbol, cognitive, networked, interactive systems with integrative representational interfaces (Engelbart 1963). Invents 'mouse' controller, windowing system and hyperlinking documents over networked system (1960–8).
	Saul Gorn redescribes computer systems with pragmatist semiotics (Gorn 1967, 1968, 1983).
	Semiotics Societies established; Semiotics as an academic field of study formally begins.

1970s	1972–7: Alan Kay, *Dynabook* concept for computers as a *metamedium*. Develops Object-Oriented Programming and symbolic interface systems at Xerox/PARC (Kay 1972, 1984; Kay and Goldberg 1977).
	1977–9: Kay co-develops Xerox PARC, Star/Alto PCs: first graphical interface, multimedia 'personal' computer systems with window layers, hypertext linking, icons and mouse pointer device. Semiotic terms – type/ token, icon, index – are adopted in GUI design at Xerox/PARC.
	ARPAnet – Internet (1970s–80s):
	Data packet protocols, implementing metadata layers and extending information theory for digital tokenization of data across networks of networks. Internet architecture and TCP/IP embodies all the requirements of a semiotic subsystem for end-to-end encoding and decoding (transmission/ reception) of all data types in the client/server architecture of the Internet.
	1970s–90s: Emerging interest in computing systems for semiotic theory (early studies by Nake, Nadin and Andersen).
1980s	1984: Apple Macintosh: consumer system version of Xerox/PARC interactive graphical system concepts (without networking).
	'Personal computer' GUI and interactive software design principles are established across operating systems, and soon become standardized for the design of all consumer and business PCs.
	Human Computer Interface design (HCI) becomes an interdisciplinary field (Shneiderman 1983, 1997; Norman and Draper 1986), combining computer science, design, programming, cognitive psychology, interaction theories and semiotic theory.
1990s	1991: Tim Berners-Lee develops the HTTP Web server system and HTML for hypertext linking among multiple networked documents as a protocol layer that uses the client/server architecture of the Internet.
	1991: Unicode founded to standardize digital code for representations of written characters of all languages.
	Digital media standards widely adopted for digitization of all symbolic types (text, graphics, image, photo, video, audio).
	1993–5: Graphical hypermedia interface software for the Web (Mosaic, Netscape).
	1994: Internet and Web opened to private development and consumer use.
	Hypertext and hypermedia systems extend semiotic principles for indexical linking with the Internet client/server architecture as a subsystem.
	HCI now includes Internet/Web multimedia design.
2000s–	Internet and Web architectures scale and extend for all digital media and information services.
	Internet-connected massively distributed computing systems (Cloud) become standard architecture.
	AI and Machine Learning (ML) methods become viable with advances in computation and massive data sets.

Research and theory on computation and information in semiotic studies: 1980s–present

There are multiple schools of thought and disciplinary contexts for semiotic theory, and, thus, for 'semiotics and computing'. The ongoing topics of research and theory in the semiotic studies community and related fields since the 1980s are summarized in Table 9.4, which presents representative work for points of entry in the field.

Several scholars have led the way in applying Peircean principles to the study of computer and information systems: the studies by Frieder Nake (1997, 2002, 2008a, 2008b); Winfried Nöth (1997, 2002); Mihai Nadin (1988b, 1998, 2007, 2011); Peter Skagestad (1993, 1996, 1999); John Sowa (1984, 1991, 2000a, 2000b) and Joseph

TABLE 9.4 Research and theory on semiotics and computing, 1980s to present.

Theories of computation: Semiosis and symbolic systems; Logic and computation; Computational theories of mind	Ketner and Stewart (1984) | Ketner (1988)
	Andersen, Holmqvist, Jensen (1993) | Skagestad (1996)
	Fetzer (1997, 2001) | Andersen (1997) | Nöth (1997, 2002)
	Andersen, Hasle, Brandt (1997) | Gudwin (1999)
	Nadin (1998, 2007, 2011) | Ransdell (2003)
	Rapaport (1999, 2012, 2018) | Sowa (2000a, 2006)
	Gomes, Gudwin, El-Hani, Queiroz (2007)
	Queiroz and Merrell (2009) | Tanaka-Ishii (2010)
Semiotic Foundations of Information Theory	MacKay (1969) | Gorn (1968, 1983) | Nadin (2011)
	Kockelman (2017a, b)
Interface Design and Interaction Programming	Engelbart (1988) | Shneiderman (1982)
	Nadin (1988b, 1988a, 2017) | Kay (2001)
	Goguen (Goguen 1999; Malcolm and Goguen 1999)
	Goguen and Harrell (2005) | De Souza (2005)
	Murray (2012)
Digital Media as a Semiotic System; Software and Digital Art	Nake (1999, 2002, 2008a, 2009; Nake and Grabowski 2006) | Murray (2012) | Manovich (2013)
Knowledge Representation, Logic, and Conceptual Structures	Sowa (1984, 1991, 2000a, 2000b)
	Meunier (1989, 1998) | Holmqvist, Klein, Posner (1996)
Semiotic Engineering Concepts	Liu (2000) | De Souza (2005) | Nadin (2017)
	Sowa (2000a, 2011) | Barbosa and Breitman (2017)
AI, Cybernetic Systems, and Semiotic Models of Data	Fetzer (Fetzer 1988, 1997, 2001)
	Skagestad (1993, 1996) | Agre (1995, 1997)
	Jorna, Van Heusden, Posner (1993) | Ketner (2003)

Goguen (Goguen 1999, 2003; Malcolm and Goguen 1999; Goguen and Harrell 2005) are foundational. Other important studies appeared in the 1990s–2000s (Fetzer 1988, 1997, 2001, 2004; Andersen et al. 1997; Queiroz and Merrell 2009). The interdisciplinary scope of semiotics and computing now also includes work in anthropology and cognitive science (see Kockelman 2005, 2006, 2010, 2017a, 2017b).

METHODOLOGIES

Describing semiotic levels in the design of digital computing systems

With the framework outlined above and the knowledge base provided by the related disciplines, we can apply a semiotic deblackboxing method for describing how and why computer systems, software and digital media are semiotic systems, designed as systems of subsystems that serve human symbolic representation, interpretation and communication. With this method, we will also be able to discover our roles as *semiotic agents*, when everything computational and digital is restored to intelligibility as a semiotic system with purposiveness, agency and interpretability built in by design. Among the fundamental principles of computing and the necessary design features of digital systems, the following topics provide useful entry points for making semiotic principles explicit in a Peircean semiotic systems description.

Computation, automation and computer systems

Our current PCs and computing devices incorporate nearly a hundred years of design solutions for technical implementations of semiotic systems.[21] We can summarize the semiotic systems view of the design solutions to the core semiotic problem. To create an electronic computer system for automating *operations* (interpretive processes) on types of symbols and produce new token instances representing interpretations, we need to design a physical system that will allow us to (1) *introduce* physical tokens of human intentional sign systems and *register* them internally in structures in the system (i.e., take in inputs through an *interface*), (2) implement *logical operations* (interpretations) assigned to those symbols as defined at other levels in the system, (3) direct the system processes to generate (retokenize) tokens for internal transitions in the operations and further tokens for what the 'input' tokens must be transformed into as a result ('outputs') of the interpretations, and then (4) project the 'output' tokens into perceptible and interpretable physical media (i.e., *interfaces*).

Further, having chosen the binary electronic architecture as the most efficient, we can use the same *binary tokenization system* to create byte representations of symbolic types (data), map the representations of human symbol systems (written characters, graphics, images, sounds) to indexed memory locations and assign to each symbolic type the pattern of interpretations (operations, relations, concept maps) that 'go with' the tokens instantiated in the system substrates.

Models of computation for digital systems have gone through several stages of development since the 1940s, each in parallel with developments in supporting subsystems (e.g., memory units, processors, displays): (1) the first systems were algorithmic calculating machines designed 'run' one terminating program at a time in a system that encoded 'symbols that mean' and 'symbols that do' in corresponding binary substrates (the finite state machine 'input-output' model, 1940s–50s), (2) computing expanded as 'general symbol processing' designed to map any encodable symbol token structure and correlated patterns of interpretive 'processing' to binary memory units and processor

structures, and output corresponding representations to displays (screens) (1960s–70s) and (3) our current paradigm of interactive programming, dialogic graphical interfaces and networked, distributed computing in many layers of software, a macro design which subsumes earlier design stages. In our interactive paradigm, system levels are combined for dialogic processes, directed by agents who can continually modify, reinterpret and create new representations (1970s–present).

Many leaders in computer science today focus the definition of computing and computer science on the implicit semiotic principles of *representations, interpretive transformations* and *information processing*, and not on computers as machines.[22] 'Representation-transformation can be a reference model of computing. An information process is a sequence of representations' (Denning 2012: 808–89). Further, the core of *computation* is not simply *representation*, but operationalizing symbolic structures so that they *cause* controlled actions as *operations*, that is, as rule-directed *interpretations* of representations. Many descriptions of computation today follow the same view of sign-actions and symbolic processes that Peirce first developed:

> Computing emphasizes the transformation of information, not simply its discovery, classification, storage, and communication. Algorithms not only read information structures, they modify them […]. [T]he structures of computing are not just descriptive, they are generative. An algorithm is not just a description of a method for solving a problem, it causes a machine to solve the problem. The computing sciences are the only sciences with such a strong emphasis on information causing action.
>
> (Denning and Martell 2105: 16–17)

These are the assumptions that enable computer *systems* to become *semiotic systems*. Assumed human cognitive-semiotic agency is 'built in' to all system levels (by assignment or delegation), and is structurally anticipated in the design principles for interactive and networked systems.

The principle of Homology: The key to the physical symbolic system

There is a logical-semiotic 'key' for understanding why and how digital electronic computing systems can be designed to both *instantiate physical symbolic representations* (tokens of symbol types) and *implement logical operations* (perform assignable interpretations and instantiate further tokens) in a unified binary architecture. The key is in how the mathematical principle of *homology* (structural correspondence mapping between domains) can be used as a *design principle*; that is, by imposing a logical map of *one-to-one correspondences* between structures in *symbolic systems* and structures in an intentionally designed *physical system*.

Peirce defined the homological mapping principle in his writings on mathematics and cartography (map-making): a *homology* (from Greek: *homo*: like, same + *logos*: structure, form, ratio, meaning) describes correspondences representing *equivalences* in *structure* or *form*. '*Homological*, having a structural affinity: distinguished from *analogical*' (1889–91: CD 2868). The principle of *one-to-one correspondence* distinguishes *homology* from simple *analogy*: '*homologous* is corresponding in a system of one-to-one correspondence' (1894–5: MS 165, NEM 2.217). 'A *correspondence* is a system of relationship between two sets of objects which connects all the objects of the first set each with the same number of objects of the second set', a definition also equivalent to an *injective function* in mathematics. *Homology* is closely parallel with the concept of

projection in geometry and map-making: '[a projection] is a system by which the points on the surface of the earth […] are made to correspond one to one to the points of a map' (1894–5: MS 94, NEM 2.286). Homologies and mapping projections are applications of the larger concept of mathematical *functions*: 'The theory of projections [… may] be said to be simply the theory of functions viewed under the strong perspective of a practical standpoint' (1889–91: CD 4763).

Digital computer systems as automated symbol processing systems would be impossible without this deeply assumed 'practical standpoint' for logical-symbolic mappings. A *homology*, used as a *design principle*, is a system of relations for *mapping* (or *projecting*) the structures of symbols and logic (represented in formal notation, graphs and diagrams) to *intentionally corresponding* physical structures, mapping one *system* onto or into another *system* (see Goguen 1999; Ambrosio 2014).[23]

Computer system architecture design provides the master plan for the homological mappings for each hardware and software subsystem so that the physical structures communicate back and forth, up and down, from and to, our input/output interfaces for interpretable token representations and communicating further semiotic agency into the system. This combination of internal and external *physical substrates* solves the core semiotic problem: how to 'realize' or 'instantiate' computations and symbolic representations in the physical affordances of the component structures (memory, processor units, user interfaces), electrical energy and time (Nisan and Schocken 2005; Denning and Martell 2015; Comer 2017; Rescorla 2017; see Table 9.5).

TABLE 9.5 Computer system homologies.

Symbolic structures	*Mapped to*	*Physical structures*
Data: 'Symbols that mean' Structures of our main symbol systems (e.g., text, image patterns) tokenized as physical patterns of bit/byte units with data *type* assignments.	⇒	**Tokenization of symbols** in substrates in long-term storage devices, and in substrates for active short-term representation arrays of binary cells in RAM memory and in processor units.
Program code: 'Symbols that do' *Metasymbolic* symbols in programs are coded for operations and interpretive processes on/for data tokens. 'Code' is also tokenized in digital bit/byte units in a program file.	⇒	**'Running code'** is projected from locations in active memory to processor units with arrays of binary logic 'gates' that *perform* operations on data tokens by first 'reading' input data tokens, and 'writing' (tokenizing) results in memory.
Formal necessity Programming code is a sign system for translating necessary relations in logic and math (represented in formal symbols and metasymbols) into binary encoded algorithms and logic in software that *anticipate interpretation* in the architecture of a computer system for *performing* computations as actions.	⇒	**Physical causality** CPUs (and clusters of processors) *translate* the binary encoded representations in programs through arrays of physical logic gates into *causal actions* (interpretive processes) over physical (tokenized) *data representations*. CPU's must also control and time-sequence operation cycles for *performing interpretive processes* over physical time and spatial memory locations.

But viewed at the software, interface and media representation levels, we only attend to the higher levels of abstraction (the 'user-facing' levels) in the telic design: the *observable* levels of outputs and inputs are directly mapped to the lower unobservable physical levels, which are designed to 'communicate up' through the system. This 'stacking' of levels enables designers, programmers and users to take 'the *logical equivalence* of hardware and software' for granted, as Saul Gorn lucidly explained (Gorn 1968). At our observable 'user' levels, physical homologies are designed to disappear into pure functionality.

Mapping principles are used in many contexts in computer system design, but viewed at a macro, unifying level, the homologous mapping principle is what enables us to create *automated computation* by translating *formal (symbolic) necessity* (represented in code) into *physical causality* (in computing components). The recognition of the formal-to-physical mapping principle, a Leibnizian 'mechanical thread', extends back to the formal symbolic logic systems developed by Peirce and his contemporaries in Boolean algebras and diagrammatic systems. The symbolic systems for formalizing necessary relations, developed from Peirce's era to Turing's in the 1930s, demonstrated that logical necessity could, in principle, be automated, provided that we can map the formal structures in a system of one-to-one correspondences for translating *formal necessity* into controlled *physical causality* (Robinson 1979; Robinson and Voronkov 2001; Rocchi 2013) (see Table 9.5). The mapping of formal *metasymbolic* structures (programming code) to physical architecture structures that perform actions is the *sine qua non* of digital electronic computation as a system of active, dynamic, interpretation processes.

Information and binary systems: Designing semiotic subsystems

'Information theory', as developed in electrical engineering, is an *engineering solution* to a *semiotic problem*: how can we impose a design on electrical current (and radio waves) for a system of *predictable* patterns that are *invariant* over places, times and material media, so that we can use the *energy patterns* to represent intentionally *meaningful patterns* in a communicable human sign system? Short answer: we can only efficiently impose this kind of controllable, predictable pattern on *switched states* of an electrical circuit: closed/open, on/off, voltage present/voltage absent. This is a binary, one-of-two-possible-states system, which maps exactly to the binary (base 2) number system and to the *logical values* in Boolean logic (T/F, yes/no). One unit of a switched state is a *bit* (binary unit); string them together in ordered patterns and we get bytes, which we can use to encode the structures of any digitized symbolic system. We implement the binary system map in matrices of miniature transistors (memory units) and chains of combined logic switches (logic 'gates' in processors). Digital information, then, is a design for a *semiotic subsystem*, a technique for tokenizing representations of *symbolic* structures in homologous physical substrates (mapping symbolic patterns to physical patterns).[24]

For an automated *electronic computational system*, then, only binary electronics allows us to create an exact system of *one-to-one correspondences*. This correspondence system allows us to (1) physically *tokenize representations* in formal-to-physical mappings in memory cells (bits and byte units: data) and (2) *perform operations on representations* by means of combinations of binary logic switches 'hard-wired' in millions of 'logic gates' in microprocessors. The combination of (1) and (2) is the definition of digital *computation* (see Table 9.5 below).

Computer system design also includes a method for managing levels and types of binary bit representations. At digital bit-level representations, both 'data' and 'program

code' are stored as binary data in memory units, but computer systems are designed to *index* and *type* memory locations so that the tokenized bits and bytes can be referenced and internally retokenized to function at their *assigned* symbolic levels. Bits/bytes only 'mean' in a *system of interpretation* represented at another level, thus demonstrating the essential semiotic structure of all things digital.

Unlike other material substrates in the history of sign systems (spoken language, traditional writing materials, image supports, analog media), digital information is *structurally semiotic* in that the subsystem requires applying abstract symbolic thought itself to *impose* a logical structure on materials and energy that are meaningless in themselves (Blanchette 2011; Abbott 2019; Patt and Patel 2020). The mapping principle followed in engineering processes for implementation in electronic components allows us to instantiate bits and bytes as *structure-preserving structures* (replicable and transmittable patterns), creating the predictable, controllable structures required for all data tokenizing systems, from what we input through our keyboards and mouse clicks to internet packets sent to initiate a remote Cloud computing data process.

Computers, symbol processing and semiotic architecture

Peirce's *semeiotic*, which includes the principles for physical symbolic homologies and the logic of operations, allows us to complete and reframe the definitions of computers as 'symbol systems'. Always aware of the necessity of physical instantiations of signs as shared cognitive anchors, Peirce also saw how sign systems, in ongoing patterns of representations and interpretations, can form semiotic 'strata' or levels of signs:

> In consequence of every sign determining an Interpretant, which is itself a sign, we have sign overlying sign. The consequence of this, in its turn, is that a sign may, in its immediate exterior, be of one of the three classes [icon, index, symbol], but may at once determine a sign of another class. But this in its turn determines a sign whose character has to be considered. This subject has to be carefully considered, and order brought into the relations of the strata of signs.
>
> (*Minute Logic*, MS 425 [1902]:134–5)

This description is an excellent starting point for understanding the sign-system levels in the physical tokenization of structures of symbolic types and the delegated interpretants in software, which combine to make a computer system a designed semiotic system. We need only add Peirce's extensive treatment of symbolic operations and interpretations to fill in the model for digital computer systems and the 'strata' of signs for digital media representations managed in the homologous maps for information, processing and interactive interface representations.

A model for 'symbol systems' emerged the 1950s–70s, which became part of the discourse in computer science and AI. Allen Newell and Herbert Simon developed the 'physical symbol system' model, which combined the computational theory of mind in cognitive science with the concepts of symbols, logic and rule-governed operations in computer science (Newell and Simon 1976; Haugeland 1981b; Simon 1996). The 'physical symbol system' descriptions get us part way to a semiotic model, but the theory is based on an impoverished conception of signs and symbols, mostly modelled on the formal symbols of symbolic logic notation, with rules for logical operations and relations, that can be assigned in the computer architecture. The role of semiotic agency and

the function of interpretant relations in a full triadic symbol system model, in Peirce's sense, are unaccounted for. However, the assumptions and terminology of the 'physical symbol system' hypothesis continue to inform arguments about symbols in theories of computation, cognition and AI (Marcus 2001; Nilsson 2007; Steels 2007; Conery 2012; Rapaport 2012).

We use the binary subsystems for byte patterns of all digitizable symbolic types and methods for representation; and, at the digital token level, a computer system is designed as a dynamic system of unlimited *retokenization*: 'tokens in' and new 'tokens out' in the managed 'strata of signs'. As Haugeland explains, using Peirce's terms, in his classic study of AI:

> A computer is an interpreted automatic formal system. […] A digital system is a set of positive and reliable techniques (methods, devices) for producing and reidentifying tokens, or configurations of tokens, from some prespecified set of types. […] Digital techniques are write/read techniques. 'Writing' a token means producing one of a given specified type (possibly complex); 'reading' a token means determining what type it is. A 'write/ read cycle' is writing a token and then (at some later time) reading it; a write/read cycle is successful if the type determined by the read technique is the same as the type specified to the write technique.
>
> (Haugeland 1985: 48, 53–4)

This is a useful general description of how all the *unobservable* symbolic homologies are designed to make what we *do* observe in our interface representations possible as components of a semiotic system.

Programming languages, code, running software and interfaces

The design history of programming languages and all that we call code is a fascinating story of applied semiotics.[25] At the beginnings of electronic computer system design and code for operations, John von Neumann (designer of the main homologous system architecture that we still use today) understood what Leibniz called 'mechanical thread', and he described the challenge of designing a code system for automated reasoning that mapped onto the state of components in the 1940s–50s:

> Our problem [for the coding of operations] is, then, to find simple, step-by-step methods […]. Since coding is not a static process of translation, but rather the technique of providing a dynamic background to control the automatic evolution of a meaning, it has to be viewed as a logical problem and one that represents a new branch of formal logics.
>
> (Goldstine and Von Neumann 1963: 83)

Peirce would have fully agreed. Of course, we now have a full suite of 'high level programming languages' (e.g., the C family, Java, Python, JavaScript), for which teams of programmers work at high levels of abstraction above the physical systems. But even though the system homologies can be forgotten because they are built in and standardized ('the *logical equivalence* of hardware and software'), coding a program 'to control the automatic evolution of a meaning' directed by semiotic agents continues to be the prime directive of coding software for computational *telic* systems.

Programming today begins by composing a 'human readable' file termed the 'source code', consisting of a metalanguage of formally specified terms, phrases and symbols for operations, relations and interpretations defined in the programming language. For the intentions encoded in the 'source code' file to 'run' in an actual computer system, the file must be translated into binary representations that can be directly installed (mapped into) computer memory and the structures of processors.[26] The diagram in Table 9.6 provides a map of the interpretive processes that enables the 'source code', written in a high-level programming language, to be mapped to the digital computer system and become part of an active system with 'users' (semiotic agents). (The table assumes the interactive software paradigm.) Each step from source code to running code requires a delegated *interpretant* system, 'meta-software' designed to translate one encoded state into another. Of course, what we code in the symbols, logic and algorithms in a 'source code' program file is as equally motivated and directed by cumulative human agency as the active software that we 'run' and interact with for all our symbolic systems encoded as types of digital media.

Interfaces as dialogic semiotic substrates, and computers as a metamedium

Material *interfaces* for enabling the symbolic pattern recognition (token → type relations) and dialogic interpretation processes with symbol representations have a deep cultural history, and digital interface design began by simulating our common two-dimensional representational substrates (surfaces for written symbols and images). Because all human sign systems must have physical-perceptible structures, symbolic structures come with *built-in interfaces* that enable inferences to systems of meanings outside physical instances. Contemporary pixel-based screens are controlled by graphics processors designed for rendering physical token structures (representations) of all our 2D symbol systems, and, by using projective geometry, for rendering simulations of 3D structures.

The designers of our interactive graphical interfaces were both applied semioticians and systems engineers. The interface design concept that began in Doug Engelbart's lab

TABLE 9.6 Programming and software: Source code to dialogic interaction.

Source Code	*Interpretant system*	*Binary 'machine code' file (or interpreted code at 'run time')*	*Interpretant system*	*'Running code' activated and directed by semiotic agents*
⇒	⇒	⇒	⇒	
Program text file (in Unicode bytecode representations) written in a high-level programming language (e.g., C++, Python, Java, etc.).	Complier program or interpreter translates source code text into binary 'machine level' code.	A binary code program file, as copied to a storage device, is assignable to a physical system as executable ('runnable') code.	Operating system 'writes' a tokenized 'copy' of the program into RAM, and CPUs initiate instructions for processes for specified data types.	'Users' are semiotic agents, dialogically interacting with the software for the symbolic systems interpreted and represented in the software, in a semiosic cycle.

and continued through all versions of windowing interfaces in PCs, distinguished three levels of 'interfaces': the *physical*, the *cognitive* and the *conceptual* (Card and Moran 1988; Moggridge 2007). To embody the interface concepts in the software behind what we see rendered in screens, graphical interfaces are designed with a 'meta' layer that we now take for granted in computer devices as two-way dialogic systems. The 'interface', as a semiotic substrate, is not simply a passive display for static representations, but incorporates an input system layer for communicating semiotic agency (intentions, choices, directions) back into the system for ongoing dialogic interaction with dynamic configurations of representations projected into the physical substrates of the screen.

Our current interface designs support the 'interactive computing paradigm', which was developed by using implicit semiotic principles for designing non-terminating programs for multi-symbolic systems and recursive dialogic interpretive processes.[27] Further, as Licklider and Kay envisioned, a digital multimedia interactive computer system is not correctly conceived as a *medium*, but as a *metamedium*, a medium for representing, interpreting, communicating and creating new instances of all symbolic media. Our contemporary computing paradigm is thus an implementation of Peirce's dialogic model for dynamic symbolic systems, which includes different kinds of semiotic agency in the new combined system of human cognizers and distributed agency in many layers of software and networked systems.

CONCLUSIONS

There can't be a 'semiotic approach' to the study of computer systems, software, digital media, interfaces or the internet because these technologies are *constitutively and structurally semiotic*. That is, digital computing and information technologies are (1) complex-system artefacts designed *by means of* the cognitive-symbolic capacities of human sign-using communities, and (2) the whole architecture of subsystems and supporting technologies follows telic design principles for *serving* human symbolic systems and their corresponding patterns and actions of interpretation. Computer system design principles provide homological maps for symbolic to physical correspondences that enact assigned representations and operations. Any computer system, large, small or unobservable, represents an *implementable design* of applied semiotic structures in a unified architecture based on, and in the service of, human symbolic thought.

From a pragmatist semiotic perspective, the *computer system* is not just the complex physical system of hardware, software and data (the hidden artefactual structures in machines, networks and stored information), even when correctly described as semiotic artefacts. The 'computer system' is actually the *whole dialogic supersystem* comprised of semiotic agents (aka 'users'), who are not independent individuals but members of meaning-making communities, and computer *systems* embodying semiotic system design for dialogic interaction. As Engelbart originally envisioned, human cognizers + dynamic computational semiotic systems form a *whole new third system* not reducible to a sum of the constituents. We are *members* and *agents* of the designed systems, *presupposed and included* in the designs, not detached, empirical observers of a 'machine' (Winograd and Flores 1987; Winograd 1997). We activate the built-in agency position in all the interactive-dialogic relations with the physical architectures, in the 'code' of any running software, in the affordances of interfaces, and in all accessed networked information, near or far.

There are many other levels and contexts of semiotic functions in the design and use of software, databases, digital media, interfaces and AI, and these applications will open up for semiotic description by extending the semiotic systems *deblackboxing method* outlined here. Further, by extending Peirce's *semeiotic* for our contemporary context, we have an open opportunity for bridge-building across disciplines, for embracing all knowledge domains relevant for semiotic research, and for reclaiming the foundational history of ideas woven with Leibniz's 'mechanical thread'.

NOTES

1 Leibniz's philosophy of symbols and the metaphor of the 'mechanical thread' are in Leibniz 1975 (Loemker, ed.) and Dascal 1987. For the texts on Leibniz's mechanical calculator, the binary ('dyadic') number system and his binary calculator, see Leibniz [1679] 2010 and [1710] 2009.

2 For background on the intellectual history and theory from different disciplinary viewpoints, see: Gorn 1968; Skagestad 1996; Andersen et al. 1997; Frank 2003; Gudwin and Queiroz 2005; Nadin 2007; Tedre 2014; Meunier 2018.

3 On 'deblackboxing', see Latour 1999: 183–93, and 2002; on closed, locked-in computing devices, see Zittrain 2009.

4 On mapping principles in diagrammatic reasoning and applications in computer systems, see Sowa 1984: 367–402; Glasgow, et al., eds. 1995; Goguen 1999; Sowa 2000a; Goguen and Harrell 2005; Stjernfelt 2007; Denning and Martell 2015: 123–35.

5 For background on the *implicit* and *explicit* semiotic foundations in the design history of computing, which includes earlier logic machines, diagrams, 'paper machines', and pre-digital methods for automating reasoning, see Gardner 1958; Webb 1980; Krämer 1988; Aspray 1990; Marciszewski and Murawski 1995; Priestley 2011; von Plato 2017, and for the intellectual history of computers as symbolic systems, see Mahoney 2011.

6 For example: Engelbart 1963; Winograd and Flores 1987; Card and Moran 1988, and other papers in Goldberg (ed) 1988; Rheingold 2000; Murray 2012; Manovich 2013; Rocchi 2013; Dasgupta 2014; Tedre 2014.

7 The following studies from various schools of thought on cognition, computation, and the computational theory of mind, include both implicit and explicit semiotic theory: Haugeland 1981a; Pylyshyn 1984; Schank and Childers 1984; Winograd and Flores 1987; Horst 1996; Agre 1997; Cummins and Cummins 2000; Scheutz 2002; Nilsson 2007; Clark 2008; Dror and Harnad 2008; Nilsson 2009; Rapaport 2012; Rescorla 2020.

8 Peirce's explicit definitions for his later program of *Logic [considered] as Semeiotic* begin in 1896 (MSS 900, 900(s), *Logic of Mathematics*) and 1897 (MSS 738 and 798, On logic and semeiotic), and he develops the theory continuously from 1901–2 (in his drafts of the *Minute Logic* project, especially MS 425, which treats 'reasoning by machinery') to his final papers in 1913, a year before his death. On the first page of notebook pages from 1903, Peirce wrote the title, *Mathematics as It Is to Be Treated in My Logic Treated as Semeiotics* (MS 66). For general background, see Fisch 1986: 338–42, Colapietro 2003, Pietarinen 2006, and Bellucci 2014. I treat Peirce's *Logic as Semeiotic* in relation to computing, information, AI and symbolic thought in a forthcoming book.

9 In my comprehensive survey of Peirce's writings from 1890–1914 (in thousands of pages of his unpublished papers, and in his published articles and recent editions), I found that Peirce uses the term, *Logic as Semeiotic* (and equivalent phrases) over fifty times. During this period,

Peirce's preferred spelling is *semeiotic*, sometimes *semiotic*, and very rarely in the plural form, *semeiotics*/*semiotics* (among over ninety uses of the terms). Peirce intended *semeiotic* to preserve the meanings in the traditions of logic (Greek: *semeiotike*), represented by the term used in works by John Locke and German logicians; but Peirce generalized *semeiotic* for formalizing the structures of all sign systems, especially the necessary structures of reasoning in mathematics and logic.

10 See: Hintikka 1996; Anellis 2015; Øhrstrom 2017; and essays in Houser et al. 1997.

11 The standard descriptions of computer system architecture and design principles are treated in all textbooks on the subject; the following provide accessible orientations: Heuring and Jordan 2003; Saltzer and Kaashoek 2009; and especially Tedre 2014 and Denning and Martell 2015. For thorough technical descriptions, see Blaauw and Brooks 1997; Comer 2017; Hennessy and Patterson 2017; Patt and Patel 2020. Valuable for systems theory concepts are Winograd and Flores 1987; Simon 1996; and Arthur 2011.

12 For an orientation to the principle of levels of abstraction, subsystems, and system design see: Simon 1996; Baldwin and Clark 2000; Floridi 2008; Gobbo and Benini 2014; Denning and Martell 2015: 198–212; Rescorla 2017.

13 Important studies that discuss or assume the cognitive artefact concept for computing, in different disciplinary contexts, are: Gorn 1968; Norman 1991; Hutchins 1999; Mahoney 2005; Houkes and Vermaas 2010; Nadin 2011; Borgo et al. 2014; Kockelman 2017b; Turner 2018; Anderson 2019; Sørensen et al. 2020.

14 Important sources are Clark and Chalmers 1998; Latour 1999: 176–98; Hollan et al. 2000; Dascal and Dror 2005; Zhang and Patel 2006; Dror 2007; Clark 2008; Dror and Harnad 2008; Enfield and Kockelman 2017; Kockelman 2017a.

15 The story of Peirce's unrecognized contributions to, and anticipations of, the foundations of modern computing has yet to be told; for intellectual historical facts, connections, and insights, see Ketner and Stewart 1984; Ketner 1988; Gandy 1995; Skagestad 1996; Nöth 1997, 2003; Nadin 2011, 2017.

16 Researchers will find the selections of papers in EP2, NEM (ed. Eisele 1985), and SWS (ed. Bellucci 2020) to be good starting points, but the most important writings from 1906–12 that apply to computation, symbolic operations, and logical machines have not been published. I provide a catalog of these papers, and edited selections of the most important sources, on my website; available: https://irvine.georgetown.domains/Peirce/.

17 The sources for these concepts in Peirce's papers are documented on my website; available:https://irvine.georgetown.domains/Peirce/.

18 The important developments in computing referenced here are documented in Ceruzzi 1983, 2003; Rheingold 2000; Ifrah 2001; Mahoney 2011; Davis 2012; and Campbell-Kelly and Aspray 2014. For the key concepts in interface and interaction design for our GUI systems from the 1960s on, see Goldberg (ed) 1988; and Moggridge 2007.

19 Leibniz 1679, 1710; 1975 (Loemker, ed); Dascal 1987.

20 Licklider 1960, 1965, 1977; Licklider and Clark 1962; Licklider and Taylor 1968; and see: Waldrop 2001.

21 Accessible and 'semiotics aware' introductions to computing and computer systems are Tedre 2014; Denning and Martell 2015. Additional useful guides for the key concepts in computation are Hilton 1963; Smith 1998, 2002; Mahoney 2011; Davis 2012. More advanced accounts of the history of logic and automated reasoning, which reveal implicit semiotic principles, are Marciszewski and Murawski 1995; Rojas and Hashagen 2000; Priestley 2011; von Plato 2017.

22 See the papers from the ACM Ubiquity Symposium on 'What Is Computation?' (2011), available online: https://ubiquity.acm.org/symposia2011.cfm. The papers were also published in *The Computer Journal* 55 (7) (2012). This approach is also followed by Tedre (2014) and Denning and Martell (2015), and assumed throughout in Rheingold (2000).

23 To avoid confusion in terminology, we need to differentiate *homology* from *analogy* (any kind of likeness or comparison) and from the related terms *isomorphism* and *hom(e)omorphism*, used for strictly defined abstract, mathematical *equivalences* (as in category theory and topology) (Krömer 2007; Marquis 2009). The term *homology* is also used in other sciences, and you may find the terms from abstract mathematics used in the computing literature for mappings in digital architecture. But *homology*, in the general sense defined by Peirce, is the most appropriate term for the one-to-one, formal-to-physical *imposed* correspondences in digital computer system design.

24 'Information Theory', as defined in electrical engineering and digital design, is continually misunderstood and mystified. For sources and useful explanations, see Shannon and Weaver 1949; Pierce 1980; Frank 2003; Gleick 2011; Nadin 2011; Denning and Martell 2015: 35–58; the implicit and explicit semiotic foundations of digital information are also discussed in MacKay 1969 and essays in Machlup and Mansfield (eds) 1983.

25 Accessible introductions are Turbak and Gifford 2008; Martin 2010; Denning and Martell 2015: 83–121.

26 The binary code translator programs are called 'compilers' and 'interpreters'.

27 Important sources for supporting a semiotic view of interactive systems are Engelbart 1963; Hutchins et al. 1985; Agre and Rosenschein 1996; Wegner 1997; De Souza 2005; Goldin et al. 2006; Moggridge 2007; Murray 2012.

REFERENCES

Abbott, R. (2019), 'The Bit (and Three Other Abstractions) Define the Borderline between Hardware and Software', *Minds and Machines*, 29/2: 239–85.

Agre, P. E. (1995), 'The Soul Gained and Lost: Artificial Intelligence as a Philosophical Project', *Stanford Humanities Review*, 4/2: 1–19.

Agre, P. E. and S. J. Rosenschein, eds (1996), *Computational Theories of Interaction and Agency*, Cambridge, MA: MIT Press.

Agre, P. E. (1997), *Computation and Human Experience*, Cambridge; New York: Cambridge University Press.

Ambrosio, C. (2014), 'Iconic Representations and Representative Practices', *International Studies in the Philosophy of Science*, 28/3: 255–75.

Andersen, P. B., B. Holmqvist, and J. F. Jensen, eds (1993), *The Computer as Medium*, Cambridge: Cambridge University Press.

Andersen, P. B., P. Hasle, and P. A. Brandt (1997), 'Machine Semiosis', in Roland Posner, Klaus Robering, and Thomas A., Sebeok (eds), *Semiotik/Semiotics: A Handbook on the Sign-Theoretic Foundations of Nature and Culture*, Vol. 1, 548–71, Berlin and New York: Mouton De Gruyter.

Anderson, N. G. (2019), 'Information Processing Artifacts', *Minds and Machines*, 29/2: 193–225.

Anellis, I. H. (2015), 'Peirce's Role in the History of Logic: Lingua Universalis and Calculus Ratiocinator', in Arnold Koslow and Arthur Buchsbaum (eds), *The Road to Universal Logic*, Vols 1–2, Vol. 2, 135–69, Cham: Birkhäuser / Springer.

Arthur, W. B. (2011), *The Nature of Technology: What It Is and How It Evolves*, New York, NY: Free Press.
Aspray, W., ed. (1990), *Computing before Computers*, Ames, IA: Iowa State Press.
Baldwin, C. Y. and K. B. Clark (2000), *Design Rules, Vol. 1: The Power of Modularity*, Cambridge, MA: MIT Press.
Barbosa, Simone Diniz Junqueira, and Karin Breitman, eds. (2017), *Conversations Around Semiotic Engineering*, Cham, Switzerland: Springer.
Bellucci, F. (2014), '"Logic, Considered as Semeiotic": On Peirce's Philosophy of Logic', *Transactions of the Charles S. Peirce Society*, 50/4: 523–47.
Blaauw, G. A. and F. P. Brooks (1997), *Computer Architecture: Concepts and Evolution*, Reading, MA: Addison-Wesley.
Blanchette, J. F. (2011), 'A Material History of Bits', *Journal of the American Society for Information Science and Technology*, 62 (6): 1042–57.
Borgo, S., M. Franssen, P. Garbacz, Y. Kitamura, R. Mizoguchi, and P. E. Vermaas (2014), 'Technical Artifacts: An Integrated Perspective', *Applied Ontology*, 9 (3–4): 217–35.
Bush, V. (1945), 'As We May Think', *The Atlantic*, July: 101–8.
Campbell-Kelly, M. and W. Aspray (2014), *Computer: A History of the Information Machine*, 3rd edn, Boulder, CO: Westview Press.
Campbell-Kelly, M. and S. B. Russ (1994), 'Computing and Computers', in Ivor Grattan-Guinness (ed.), *Companion Encyclopedia of the History and Philosophy of the Mathematical Sciences*, Vols. 1 and 2, 701–7, London; New York: Routledge.
Card, S. K. and T. P. Moran (1988), 'User Technology–from Pointing to Pondering', in Adele Goldberg (ed.), *A History of Personal Workstations*, 489–521, New York: ACM Press and Addison-Wesley.
Ceruzzi, P. E. (1983), *Reckoners: The Prehistory of the Digital Computer, from Relays to the Stored Program Concept*, 1935–45, Westport, CT: Greenwood.
Ceruzzi, P. E. (2003), *A History of Modern Computing*, 2nd edn, Cambridge, MA: MIT Press.
Cherry, C. (1957), *On Human Communication: A Review, a Survey, and a Criticism*, Cambridge, MA: MIT Press.
Clark, A. (2008), *Supersizing the Mind: Embodiment, Action, and Cognitive Extension*, New York, NY: Oxford University Press.
Clark, A. and D. Chalmers (1998), 'The Extended Mind', *Analysis*, 58 (1): 7–19.
Colapietro, V. (2003), 'The Space of Signs: C. S. Peirce's Critique of Psychologism', in Dale Jacquette (ed.), *Philosophy, Psychology, and Psychologism: Critical and Historical Readings on the Psychological Turn in Philosophy*, 157–80, Dordrecht; Boston: Kluwer.
Comer, D. E. (2017), *Essentials of Computer Architecture*, 2nd edn, Boca Raton: CRC Press.
Conery, J. S. (2012), 'Computation Is Symbol Manipulation', *The Computer Journal*, 55 (7): 814–6.
Cummins, R. and D. D. Cummins, eds (2000), *Minds, Brains, and Computers: An Historical Introduction to the Foundations of Cognitive Science*, Malden, MA: Wiley-Blackwell.
Dascal, M. (1987), *Leibniz – Language, Signs and Thought: A Collection of Essays*, Amsterdam; Philadelphia: John Benjamins.
Dascal, M. and I. E. Dror (2005), 'The Impact of Cognitive Technologies: Towards a Pragmatic Approach', *Pragmatics & Cognition*, 13 (3): 451–7.
Dasgupta, S. (2014), *It Began with Babbage: The Genesis of Computer Science*, Oxford, UK: Oxford University Press.
Davis, M. (2012), *The Universal Computer: The Road from Leibniz to Turing*, Boca Raton, FL: CRC Press.

De Souza, C. S. (2005), *The Semiotic Engineering of Human-Computer Interaction*, Cambridge, MA: MIT Press.

Denning, P. J. (2012), 'Opening Statement: What Is Computation?', *The Computer Journal*, 55 (7): 805–10.

Denning, P. J. and C. H. Martell (2015), *Great Principles of Computing*, Cambridge, MA: MIT Press.

Dror, I. E. (2007), *Cognitive Technologies and the Pragmatics of Cognition*, Amsterdam; Philadelphia: John Benjamins.

Dror, I. E. and S. R. Harnad, eds (2008), *Cognition Distributed: How Cognitive Technology Extends Our Minds*, Amsterdam; Philadelphia: John Benjamins.

Eisele, C. (1979), *Studies in the Scientific and Mathematical Philosophy of Charles S. Peirce*, ed. R. M. Martin, The Hague: Mouton De Gruyter.

Eisele, C., ed. (1985), *Historical Perspectives on Peirce's Logic of Science: Parts I and II*, Vols 1–2, Berlin; New York: De Gruyter Mouton.

Enfield, N. J. and P. Kockelman, eds (2017), *Distributed Agency*, Oxford; New York: Oxford University Press.

Engelbart, D. C. (1963), 'A Conceptual Framework for the Augmentation of Man's Intellect', in Paul W. Howerton and David C. Weeks (eds), *Vistas in Information* Handling, *Volume I: The Augmentation of Man's Intellect by Machine*, 1–29, Washington, DC: Spartan Books.

Engelbart, D. C. (1988), 'The Augmented Knowledge Workshop', in Adele Goldberg (ed.), *A History of Personal Workstations*, 185–248, New York: ACM.

Eschbach, A. and J. Trabant, eds (1983), *History of Semiotics*, Amsterdam; Philadelphia: John Benjamins.

Fetzer, J. H. (1988), 'Signs and Minds: An Introduction to the Theory of Semiotic Systems', in James H. Fetzer (ed.), *Aspects of Artificial Intelligence*, 133–61, Dordrecht & Boston: Kluwer.

Fetzer, J. H. (1997), 'Thinking and Computing: Computers as Special Kinds of Signs', *Minds and Machines*, 7 (3): 345–64.

Fetzer, J. H. (2001), *Computers and Cognition: Why Minds Are Not Machines*, Dordrecht: Springer.

Fetzer, J. H. (2004), 'Peirce and the Philosophy of Artificial Intelligence', in M. Bergman and J. Queiroz (eds), *The Commens Encyclopedia: The Digital Encyclopedia of Peirce Studies*, New edn, http://www.commens.org/encyclopedia/article/fetzer-james-peirce-and-philosophy-artificial-intelligence.

Fisch, M. H. (1986), *Peirce, Semeiotic and Pragmatism*, Bloomington: Indiana University Press.

Floridi, L. (2008), 'The Method of Levels of Abstraction', *Minds and Machines*, 18 (3): 303–29.

Frank, H. (2003), 'Semiotik und Informationstheorie', in Roland Posner, Klaus Robering, and Thomas A. Sebeok (eds), *Semiotik/Semiotics: A Handbook on the Sign-Theoretic Foundations of Nature and Culture*, Vol. 3, 2418–37, Berlin and New York: Mouton De Gruyter.

Gabbay, D. M. and J. Woods, eds. (2004), *Handbook of the History of Logic, Vol. 3: The Rise of Modern Logic from Leibniz to Frege*, Amsterdam; Boston: North Holland.

Gabbay, D. M., J. H. Siekmann, and J. Woods, eds. (2014), *Handbook of the History of Logic, Vol. 9: Computational Logic*, Amsterdam: North Holland.

Gandy, R. (1995), 'The Confluence of Ideas in 1936', in Rolf Herken (ed.), *The Universal Turing Machine: A Half-Century Survey*, 50–102, Wien; New York: Springer.

Gardner, M. (1958), *Logic Machines and Diagrams*, New York: McGraw–Hill.

Glasgow, J., N. Hari, and B. Chandrasekaran, eds (1995), *Diagrammatic Reasoning: Cognitive and Computational Perspectives*, Cambridge, MA: AAAI Press.

Gleick, J. (2011), *The Information: A History, a Theory, a Flood*, New York, NY: Pantheon.
Gobbo, F. and M. Benini (2014), 'The Minimal Levels of Abstraction in the History of Modern Computing', *Philosophy & Technology*, 27 (3): 327–43.
Goguen, J. A. (1999), 'An Introduction to Algebraic Semiotics, with Application to User Interface Design', in Chrystopher L. Nehaniv (ed.), *Computation for Metaphors, Analogy, and Agents*, 242–91, Berlin & Heidelberg: Springer.
Goguen, J. A. (2003), 'Semiotic Morphisms, Representations and Blending for Interface Design', in F. Spoto, G. Scollo, and A. Nijholt (eds), *Algebraic Methods in Language Processing*, Vol. 21, 1–15, Twente: University of Twente.
Goguen, J. A. and D. F. Harrell (2005), 'Information Visualization and Semiotic Morphisms', in Grant Malcolm (ed.), *Multidisciplinary Approaches to Visual Representations and Interpretations*, Vol. 2, 83–98, Amsterdam; London: Elsevier.
Goldberg, A., ed. (1988), *A History of Personal Workstations*, New York: ACM Press; Addison-Wesley.
Goldin, D., S. A. Smolka, and P. Wegner, eds (2006), *Interactive Computation: The New Paradigm*, Berlin; New York: Springer.
Goldstine, H. H. and J. Von Neumann (1963), 'Planning and Coding Problems for an Electronic Computing Instrument, Part II, Vol. 1 (1946)', in A.H. Taub (ed.), *Collected* Works, *Vol. 5: Design of Computers, Theory of Automata and Numerical Analysis*, Vol. 5, 80–151, Oxford; London; New York: Pergamon Press.
Gomes, A., R. Gudwin, C. N. El-Hani, and J. Queiroz (2007), 'Towards the Emergence of Meaning Processes in Computers from Peircean Semiotics', *Mind & Society*, 6 (2): 173–87.
Gorn, S. (1967), 'The Computer and Information Sciences and the Community of Disciplines', *Behavioral Science*, 12 (6): 433–52.
Gorn, S. (1968), 'The Identification of the Computer and Information Sciences: Their Fundamental Semiotic Concepts and Relationships', *Foundations of Language*, 4 (4): 339–72.
Gorn, S. (1983), 'Informatics (Computer and Information Science): Its Ideology, Methodology, and Sociology', in Fritz Machlup and Una Mansfield (eds), *The Study of Information: Interdisciplinary Messages*, 121–40, New York: Wiley-Interscience.
Grier, D. A. (2005), *When Computers Were Human*, Princeton: Princeton University Press.
Gudwin, R. R. (1999), 'Umwelts and Artificial Devices: A Reflection on the Text of Claus Emeche: Does a Robot have an Umwelt ?' Seminário Avançado de Comunicação e Semiótica 2: 51–6.
Gudwin, R. and J. Queiroz (2005), 'Towards an Introduction to Computational Semiotics', in *International Conference on Integration of Knowledge Intensive Multi-Agent Systems* (KIMAS 2005), 393–8, Waltham, MA: IEEE.
Haugeland, J., ed. (1981a), *Mind Design: Philosophy Psychology Artificial Intelligence*, Cambridge, MA: MIT Press.
Haugeland, J., ed. (1981b), 'Semantic Engines: An Introduction to Mind Design', in *Mind Design: Philosophy Psychology Artificial Intelligence*, 2–33, Cambridge, MA: MIT Press.
Haugeland, J. (1985), *Artificial Intelligence: The Very Idea*. Cambridge, MA: MIT Press.
Hennessy, J. L. and D. A. Patterson (2017), *Computer Architecture: A Quantitative Approach*, 6th edn, Cambridge, MA: Morgan Kaufmann.
Heuring, V. P. and H. F. Jordan (2003), *Computer Systems Design and Architecture*, 2nd edn, Upper Saddle River, N.J: Pearson.
Hilton, A. M. (1963), *Logic, Computing Machines, and Automation*, Cleveland, OH: Meridian Books.

Hintikka, J. (1996), 'The Place of C. S. Peirce in the History of Logical Theory', in *Lingua Universalis vs. Calculus Ratiocinator: An Ultimate Presupposition of Twentieth-Century Philosophy*, 140–61, Dordrecht: Kluwer.

Hollan, J., E. Hutchins, and D. Kirsh (2000), 'Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research', *ACM Transactions, Computer-Human Interaction*, 7 (2): 174–96.

Horst, S. W. (1996), *Symbols, Computation, and Intentionality: A Critique of the Computational Theory of Mind*, Berkeley: University of California Press.

Houkes, W. and P. E. Vermaas (2010), *Technical Functions: On the Use and Design of Artefacts*, Dordrecht; New York: Springer.

Houser, N., D. D. Roberts, and J. Van Evra, eds (1997), *Studies in the Logic of Charles Sanders Peirce*, Bloomington: Indiana University Press.

Hutchins, E. L. (1999), 'Cognitive Artifacts', in Robert A. Wilson and Frank Keil (eds), *The MIT Encyclopedia of the Cognitive Sciences*, 126–8, Cambridge, MA: MIT Press.

Hutchins, E. L., J. D. Hollan, and D. A. Norman (1985), 'Direct Manipulation Interfaces', *Human-Computer Interaction*, 1 (4): 311–38.

Ifrah, G. (2001), *The Universal History of Computing: From the Abacus to the Quantum Computer*, New York: Wiley.

Jorna, R. J., B. Van Heusden, and R. Posner, eds. (1993), *Signs, Search and Communication: Semiotic Aspects of Artificial Intelligence*, Berlin; New York: De Gruyter.

Kay, A. (1972), *A Personal Computer for Children of All Ages*, Palo Alto, CA: Xerox PARC.

Kay, A. (1984), 'Computer Software', *Scientific American*, 251 (3): 52–9.

Kay, A. (2001), 'User Interface: A Personal View', in Randall Packer and Ken Jordan (eds), *Multimedia: From Wagner to Virtual Reality*, 121–31, New York: W. W. Norton.

Kay, A. and A. Goldberg (1977), 'Personal Dynamic Media', *Computer*, 10 (3): 31–41.

Ketner, K. L. and A. F. Stewart (1984), 'The Early History of Computer Design: Charles Sanders Peirce and Marquand's Logical Machines', *The Princeton University Library Chronicle*, 45 (3): 187–225.

Ketner, K. L. (1988), 'Peirce and Turing: Comparisons and Conjectures', *Semiotica*, 68 (1/2): 33–61.

Kockelman, P. (2005), 'The Semiotic Stance', *Semiotica*, 2005 (157): 233–304.

Kockelman, P. (2006), 'Residence in the World: Affordances, Instruments, Actions, Roles, and Identities', *Semiotica*, 2006 (162): 19–71.

Kockelman, P. (2010), 'Semiotics: Interpretants, Inference, and Intersubjectivity', in Ruth Wodak, Barbara Johnstone, and Paul E. Kerswill (eds), *The SAGE Handbook of Sociolinguistics*, 165–78, Thousand Oaks, CA: SAGE Publications.

Kockelman, P. (2017a), 'Semiotic Agency', in N. J. Enfield and Paul Kockelman (eds), *Distributed Agency*, 25–38, Oxford; New York: Oxford University Press.

Kockelman, P. (2017b), *The Art of Interpretation in the Age of Computation*, New York: Oxford University Press.

Krämer, S. (1988), *Symbolische Maschinen: Die Idee Der Formalisierung in Geschichtlichem Abriss*, Darmstadt: Wissenschaftliche Buchgesellschaft.

Krömer, R. (2007), *Tool and Object: A History and Philosophy of Category Theory*, Basel; Boston: Birkhäuser.

Latour, B. (1999), *Pandora's Hope: Essays on the Reality of Science Studies*, Cambridge, MA: Harvard University Press.

Latour, B. (2002), 'Morality and Technology', *Theory, Culture & Society*, 19 (5–6): 247–60.

Leibniz, G. W. ([1679] 2010), 'De Progressione Dyadica (Hannover MS), with Facsimile and Translation (Fr.)', (Y. Serra, ed.) https://archive.org/details/69LeibnizDiadica.
Leibniz, G. W. ([1710] 2009), 'La machine arithmétique de Leibniz (1710) (with Illustrations)', (Y. Serra and C.C. Adam, eds.) https://archive.org/details/41Leibniz.
Leibniz, G. W. (1975), *Philosophical Papers and Letters: A Selection*, 2nd edn, ed. and trans. L. E. Loemker, Dordrecht; Boston: Kluwer.
Licklider, J. C. R. (1960), 'Man-Computer Symbiosis', *IRE Transactions on Human Factors in Electronics*, HFE-1/1: 4–11.
Licklider, J. C. R. (1965), *Libraries of the Future*, Cambridge, MA: MIT Press.
Licklider, J. C. R. and W. E. Clark (1962), 'On-Line Man-Computer Communication', in *Proceedings of the May 1–3,1962, Spring Joint Computer Conference*, 113–28, New York: ACM.
Licklider, J. C. R. (1968), 'Computer Graphics as a Medium of Artistic Expression', in Metropolitan Museum of Art (ed.), *Computers and Their Potential Applications in Museums*, 273–301, New York: Arno Press.
Licklider, J. C. R. and R. W. Taylor (1968), 'The Computer as a Communication Device', *Science and Technology*, 76: 21–38.
Licklider, J. C. R. (1977), 'User-Oriented Interactive Computer Graphics', in Proceedings of the ACM/SIGGRAPH *Workshop on User-oriented Design of Interactive Graphics Systems*, 89–96, New York: ACM.
Liu, K. (2000), *Semiotics in Information Systems Engineering*, Cambridge; New York: Cambridge University Press.
Machlup, F. and U. Mansfield, eds (1983), *The Study of Information: Interdisciplinary Messages*, New York: Wiley-Interscience.
MacKay, D. M. (1969), *Information, Mechanism and Meaning*, Cambridge, MA.: MIT Press.
Mahoney, M. S. (2005), 'The Histories of Computing(s)', *Interdisciplinary Science Reviews*, 30 (2): 119–35.
Mahoney, M. S. (2011), *Histories of Computing*, Cambridge, MA: Harvard University Press.
Malcolm, G. and J. A. Goguen (1999), 'Signs and Representations: Semiotics for User Interface Design', in Ray Paton and Irene Neilsen (eds), *Visual Representations and Interpretations*, 162–72, London; New York: Springer.
Manovich, L. (2013), *Software Takes Command: Extending the Language of New Media*, London: Bloomsbury.
Marciszewski, W. and R. Murawski (1995), *Mechanization of Reasoning in a Historical Perspective*, Amsterdam; Atlanta: Rodopi.
Marcus, G. F. (2001), *The Algebraic Mind: Integrating Connectionism and Cognitive Science*, Cambridge, MA: MIT Press.
Marquis, J. P. (2009), *From a Geometrical Point of View: A Study of the History and Philosophy of Category Theory*, Dordrecht: Springer.
Martin, J. (2010), *Introduction to Languages and the Theory of Computation*, 4th edn, New York, NY.: McGraw-Hill.
Meunier, J.-G. (1989), 'Artificial Intelligence and Sign Theory', *Semiotica*, 77 (1–3): 43–64.
Meunier, J.-G. (1998), 'Categorial Structure of Iconic Languages', *Theory and Psychology*, 8 (6): 805–27.
Meunier, J.-G. (2018), 'Vers une Sémiotique Computationnelle?', *Applied Semiotics / Sémiotique appliquée*, 26: 76–107.
Moggridge, B. (2007), *Designing Interactions*, Cambridge, MA: MIT Press.

Morris, C. W. (1938), *Foundations of the Theory of Signs*, Chicago, IL: University of Chicago Press.
Morris, C. W. (1946), *Signs, Language and Behavior*, New York, NY: Prentice Hall.
Morris, C. W. (1964), *Signification and Significance: A Study of the Relations of Signs and Values*, Cambridge, MA: MIT Press.
Murray, J. H. (2012), *Inventing the Medium: Principles of Interaction Design as a Cultural Practice*, Cambridge, MA: MIT Press.
Nadin, M. (1988a), 'Interface Design: A Semiotic Paradigm', *Semiotica*, 69 (3/4): 269–302.
Nadin, M. (1988b), 'Interface Design and Evaluation: Semiotic Implications', in H. Rex Hartson and Deborah Hix (eds), *Advances in Human-Computer Interaction*, Vol. 2, 45–100, Norwood, N.J: Ablex.
Nadin, M. (1998), 'Computer', in Paul Bouissac (ed.), *Encyclopedia of Semiotics (Online)*, Oxford: Oxford University Press.
Nadin, M. (2007), 'Semiotic Machine', *The Public Journal of Semiotics*, 1 (1): 57–75.
Nadin, M. (2011), 'Information and Semiotic Processes: The Semiotics of Computation', *Cybernetics & Human Knowing*, 18 (1–2): 153–75.
Nadin, M. (2017), 'Semiotic Engineering – An Opportunity or an Opportunity Missed?', in Simone Diniz Junqueira Barbosa and Karin Breitman (eds), *Conversations Around Semiotic Engineering*, 41–63, Cham, Switzerland: Springer.
Nake, F. (1997), 'Der semiotische Charakter der informatischen Gegenstände', *Semiosis*, 85–90: 24–35.
Nake, F. (1999), 'Bildgeschichten aus Zahlen und Zufall: Betrachtungen zur Computerkunst', in Andreas Dress and Gottfried Jäger (eds), *Visualisierung in Mathematik, Technik und Kunst: Grundlagen und Anwendungen*, 117–36, Wiesbaden: Vieweg+Teubner.
Nake, F. (2002), 'Werk, *Kunstwerk, Information, Zeichen*', *in* Udo Bayer and Karl Gfesser (*eds*), *Kontinuum der Zeichen: Elisabeth Walther-Bense und die Semiotik*, 9–13, Stuttgart: J.B. Metzler.
Nake, F. and S. Grabowski (2006), 'The Interface as Sign and as Aesthetic Event', in Paul A. Fishwick (ed.), *Aesthetic Computing*, 53–70, Cambridge, MA: MIT Press.
Nake, F. (2008a), 'Surface, Interface, Subface: Three Cases of Interaction and One Concept', in Uwe Seifert and et al. (eds), *Paradoxes of Interactivity: Perspectives for Media Theory, Human-Computer Interaction, and Artistic Investigations*, 92–109, Berlin: Boston: Transcript Verlag.
Nake, F. (2008b), 'Work, Design, Computers, Artifacts', in Thomas Binder, Jonas Löwgren, and Lone Malmborg (eds), *(Re)Searching the Digital Bauhaus*, 309–31, London: Springer.
Nake, F. (2009), 'The Semiotic Engine: Notes on the History of Algorithmic Images in Europe', *Art Journal*, 68 (1): 76–89.
Newell, A. and H. A. Simon (1961), 'Computer Simulation of Human Thinking', *Science*, 134 (3495): 2011–7.
Newell, A. and H. A. Simon (1972), *Human Problem Solving*, Englewood Cliffs, NJ: Prentice Hall.
Newell, A. and H. A. Simon (1976), 'Computer Science as Empirical Inquiry: Symbols and Search', *Communications of the ACM*, 19 (3): 113–26.
Newell, A. (1980), 'Physical Symbol Systems', *Cognitive Science*, 4 (2): 135–83.
Newell, A. (1986), 'The Symbol Level and the Knowledge Level', in Zenon W. Pylyshyn and William Demopoulos (eds), *Meaning and Cognitive Structure: Issues in the Computational Theory of Mind*, Norwood, N.J: Praeger.

Newell, A. and H. A. Simon (2003), 'Symbol Manipulation', in *Encyclopedia of Computer Science*, 1731–5, Chichester, UK: John Wiley and Sons.

Nilsson, N. J. (2007), 'The Physical Symbol System Hypothesis: Status and Prospects', in Max Lungarella, Fumiya Iida, Josh Bongard, and Rolf Pfeifer (eds), *50 Years of Artificial Intelligence*, 9–17, Berlin & Heidelberg: Springer.

Nilsson, N. J. (2009), *The Quest for Artificial Intelligence: A History of Ideas and Achievements*, Cambridge, UK: Cambridge University Press.

Nisan, N. and S. Schocken (2005), *The Elements of Computing Systems: Building a Modern Computer from First Principles*, Cambridge, MA: MIT Press.

Norman, D. A. (1991), 'Cognitive Artifacts', in John M. Carroll (ed.), *Designing Interaction*, 17–38, New York, NY: Cambridge University Press.

Norman, D. A. and S. W. Draper, eds (1986), *User Centered System Design: New Perspectives on Human-computer Interaction*, Hillsdale, NJ: Lawrence Erlbaum Associates.

Nöth, W. (1997), 'Representation in Semiotics and in Computer Science', *Semiotica*, 115 (3/4): 203–13.

Nöth, W. (2002), 'Semiotic Machines', *Cybernetics & Human Knowing*, 9 (1): 5–21.

Nöth, W. (2003), 'Semiotic Machines', *SEED Journal*, 3 (3).

Øhrstrom, P. (2017), 'C. S. Peirce', in Alex Malpass and Marianna Antonutti Marfori (eds), *The History of Philosophical and Formal Logic: From Aristotle to Tarski*, 165–81, London; New York: Bloomsbury.

Patt, Y. and S. Patel (2020), *Introduction to Computing Systems: From Bits & Gates to C/C++ & Beyond*, 3rd edn, New York, NY: McGraw-Hill.

Peirce, C. S. ([1857–92] 1982–2010), Writings of Charles S. Peirce, 7 vols. (1–6, 8), Peirce Edition Project (eds), Bloomington: Indiana University Press. Cited as W.

Peirce, C. S. ([1857–1914] 1787–1951), The Charles S. Peirce Papers Harvard University, Houghton Library, MS Am 1632. Individual papers are referenced by manuscript number in R. Robin (ed.), *Annotated Catalogue of the Papers of Charles S. Peirce*, Amherst: University of Massachusetts Press, 1967, and in Robin, 'The Peirce Papers: A Supplementary Catalogue', *Transactions of the Charles S. Peirce Society* 7, 1971: 37–57. Cited as MS.

Peirce, C. S. ([1866–1913] 1976), *The New Elements of Mathematics*, 4 vols., C. Eisele (ed.), The Hague: Mouton Press. Cited as NEM.

Peirce, C. S. (1889–91), entries in *The Century Dictionary: An Encyclopedic Lexicon of the English Language*, W. D. Whitney (ed.), New York: Century Co. Cited as CD.

Peirce, C. S. ([1893–1913] 1998), *The Essential Peirce: Selected Philosophical Writings*, vol. 2, eds Peirce Edition Project, Bloomington: Indiana University Press. Cited as EP 2.

Peirce, Charles S. ([1894–1912] 2020), *Selected Writings on Semiotics, 1894–1912*, ed. F. Bellucci, Berlin: Mouton De Gruyter. Cited as SWS.

Pierce, J. R. (1980), *An Introduction to Information Theory: Symbols, Signals and Noise*, 2nd edn, first publ. 1961. New York: Dover.

Pietarinen, A. V. (2006), *Signs of Logic: Peircean Themes on the Philosophy of Language, Games, and Communication*, Dordrecht; London: Springer.

Posner, R., H. Klein, P. B. Andersen, and B. Holmqvist (1996), *Signs of Work: Semiosis and Information Processing in Organisations*. Berlin: De Gruyter.

Posner, R., K. Robering, and T. A. Sebeok, eds. (1997–2003), *Semiotik/Semiotics: A Handbook on the Sign-Theoretic Foundations of Nature and Culture*, Vols 1–3, Berlin; New York: Mouton De Gruyter.

Priestley, M. (2011), *A Science of Operations: Machines, Logic and the Invention of Programming*, New York; London: Springer.

Ransdell, Joseph (2003), 'The Relevance of Peircean Semiotic to Computational Intelligence Augmentation', *SEED Journal*, 3 (3): 5–36.

Pylyshyn, Z. W. (1984), *Computation and Cognition: Toward a Foundation for Cognitive Science*, Cambridge, MA: MIT Press.

Queiroz, J. and F. Merrell (2009), 'On Peirce's Pragmatic Notion of Semiosis: A Contribution for the Design of Meaning Machines', *Minds & Machines*, 19 (1): 129–43.

Rapaport, W. J. (1999), 'Implementation is Semantic Interpretation', *The Monist*, 82 (1): 109–30.

Rapaport, W. J. (2012), 'Semiotic Systems, Computers, and the Mind: How Cognition Could Be Computing', *International Journal of Signs and Semiotic Systems*, 2 (1): 32–71.

Rapaport, W. J. (2018), 'What Is a Computer? A Survey', *Minds and Machines*, 28 (3): 385–426.

Rescorla, M. (2017), 'Levels of Computational Explanation', in Thomas M. Powers (ed.), *Philosophy and Computing: Essays in Epistemology, Philosophy of Mind, Logic, and Ethics*, 5–28, Cham, CH: Springer.

Rescorla, M. (2020), 'The Computational Theory of Mind', in Edward N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy*, Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/entries/computational-mind/.

Rheingold, H. (2000), *Tools for Thought: The History and Future of Mind-Expanding Technology*, revised edn. Cambridge, MA: MIT Press.

Robinson, J. A. (1979), *Logic: Form and Function: The Mechanization of Deductive Reasoning*, Edinburgh: Edinburgh University Press.

Robinson, J. A. and A. Voronkov, eds (2001), *Handbook of Automated Reasoning*, Vols 1–2 vols. Amsterdam & Cambridge, MA: North Holland; MIT Press.

Rocchi, P. (2013), *Logic of Analog and Digital Machines*, Hauppauge, NY: Nova Science.

Rojas, R. and U. Hashagen, eds (2000), *The First Computers: History and Architectures*, Cambridge, MA: MIT Press.

Saltzer, J. H. and M. F Kaashoek (2009), *Principles of Computer System Design: An Introduction*, Burlington, MA: Morgan Kaufmann.

Schank, R. C. and P. G. Childers (1984), *The Cognitive Computer: On Language, Learning, and Artificial Intelligence*, Reading, MA: Addison-Wesley.

Scheutz, M., ed. (2002), *Computationalism: New Directions*, Cambridge, MA: MIT Press.

Shannon, C. E. (1938), 'A Symbolic Analysis of Relay and Switching Circuits', *Transactions of the American Institute of Electrical Engineers*, 57 (12): 713–23.

Shannon, C. E. (1948), 'A Mathematical Theory of Communication', *The Bell System Technical Journal*, 27: 379–423, 623–56.

Shannon, C. E. and W. Weaver (1949), *The Mathematical Theory of Communication*, Champaign, IL: University of Illinois.

Shneiderman, B. (1982), 'The Future of Interactive Systems and the Emergence of Direct Manipulation', *Behaviour & Information Technology*, 1 (3): 237–56.

Shneiderman, B. (1983), 'Direct Manipulation: A Step Beyond Programming Languages', *IEEE Computer*, 16 (8): 57–69.

Shneiderman, B. (1997), *Designing the User Interface*, 3rd edn, Reading, MA: Addison Wesley.

Simon, H. A. (1993), 'The Human Mind: The Symbolic Level', *Proceedings of the American Philosophical Society*, 137 (4): 638–47.

Simon, H. A. (1996), *The Sciences of the Artificial*, 3rd edn, first published, 1969. Cambridge, MA: MIT Press.

Skagestad, P. (1993), 'Thinking with Machines: Intelligence Augmentation, Evolutionary Epistemology, and Semiotic', *Journal of Social and Evolutionary Systems*, 16 (2): 157–80.
Skagestad, P. (1996), 'The Mind's Machines: The Turing Machine, the Memex, and the Personal Computer', *Semiotica*, 111 (3/4): 217–43.
Skagestad, P. (1999), 'Peirce's Inkstand as an External Embodiment of Mind', *Transactions of the Charles S. Peirce Society*, 35 (3): 551–61.
Smith, B. C. (1998), *On the Origin of Objects*, Cambridge, MA: MIT Press.
Smith, B. C. (2002), 'The Foundations of Computing', in Matthias Scheutz (ed.), *Computationalism: New Directions*, 23–58, Cambridge, MA: MIT Press.
Sørensen, B., T. Thellefsen, and M. Thellefsen (2020), 'A Peircean Semiotics of Technological Artefacts', in Tony Jappy (ed.), *The Bloomsbury Companion to Contemporary Peircean Semiotics*, 253–75, London: Bloomsbury.
Sowa, J. F. (1984), *Conceptual Structures: Information Processing in Mind and Machine*, Reading, MA: Addison-Wesley.
Sowa, J. F. ed. (1991), *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, San Mateo, CA: Morgan Kaufmann.
Sowa, J. F. (2000a), *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Pacific Grove: Brooks/Cole; Thomson.
Sowa, J. F. (2000b), 'Ontology, Metadata, and Semiotics', in Bernhard Ganter and Guy W. Mineau (eds), *Conceptual Structures: Logical, Linguistic, and Computational Issues: 8th International Conference on Conceptual Structures*, ICCS 2000, 55–81, Berlin; New York: Springer.
Sowa, J. F. (2006), 'Conceptual Graphs', in Peter Bernus, Kai Mertins, and Günter J. Schmidt (eds), *Handbook on Architectures of Information Systems*, 295–320, Berlin; New York: Springer.
Sowa, J. F. (2011), 'Future Directions for Semantic Systems', in Andreas Tolk and Lakhmi C. Jain (eds), *Intelligence-Based Systems Engineering*, 23–48, Berlin; Heidelberg: Springer.
Steels, L. (2007), 'Fifty Years of AI: From Symbols to Embodiment – and Back', in Max Lungarella, Fumiya Iida, Josh Bongard, and Rolf Pfeifer (eds), *50 Years of Artificial Intelligence*, 18–28, Berlin & Heidelberg: Springer.
Stjernfelt, F. (2007), *Diagrammatology: An Investigation on the Borderlines of Phenomenology, Ontology, and Semiotics*. Berlin: Springer.
Sutherland, I. E. (1963), 'Sketchpad: A Man-Machine Graphical Communication System', in Proceedings of the May 21–3,1963, *Spring Joint Computer Conference*, 329–46, New York, NY, USA: ACM.
Tanaka-Ishii, K. (2010), *Semiotics of Programming*, New York: Cambridge University Press.
Tedre, M. (2014), *The Science of Computing: Shaping a Discipline*, Boca Raton: CRC Press.
Turbak, F. and D. Gifford (2008), *Design Concepts in Programming Languages*, Cambridge, MA: MIT Press.
Turing, A. M. (1937), 'On Computable Numbers, with an Application to the Entscheidungsproblem', *Proceedings of the London Mathematical Society*, Series 2, 42 (1): 230–65.
Turner, R. (2018), *Computational Artifacts: Towards a Philosophy of Computer Science*, Berlin: Springer.
Von Neumann, J. (1987), 'First Draft of a Report on the EDVAC (1945)', in William Aspray and Arthur W. Burks (eds), *Papers of John von Neumann on Computers and Computing Theory*, 17–82, Cambridge, MA: MIT Press.

von Plato, J. (2017), *The Great Formal Machinery Works: Theories of Deduction and Computation at the Origins of the Digital Age*, Princeton: Princeton University Press.
Waldrop, M. M. (2001), *The Dream Machine: J.C.R. Licklider and the Revolution That Made Computing Personal*, New York, NY: Viking Penguin.
Webb, J. C. (1980), *Mechanism, Mentalism and Metamathematics: An Essay on Finitism*, Dordrecht: Reidel.
Wegner, P. (1997), 'Why Interaction Is More Powerful Than Algorithms', *Communications of the ACM*, 40 (5): 80–91.
Winograd, T. (1997), 'The Design of Interaction', in Peter J. Denning and Robert M. Metcalfe (eds), *Beyond Calculation: The Next Fifty Years of Computing*, 149–62, New York: Springer.
Winograd, T. and F. Flores (1987), *Understanding Computers and Cognition: A New Foundation for Design*, Reading, MA: Addison-Wesley.
Zhang, J. and V. L. Patel (2006), 'Distributed Cognition, Representation, and Affordance', *Pragmatics & Cognition*, 14 (2): 333–41.
Zittrain, J. (2009), *The Future of the Internet – And How to Stop It*, New Haven, CT: Yale University Press.