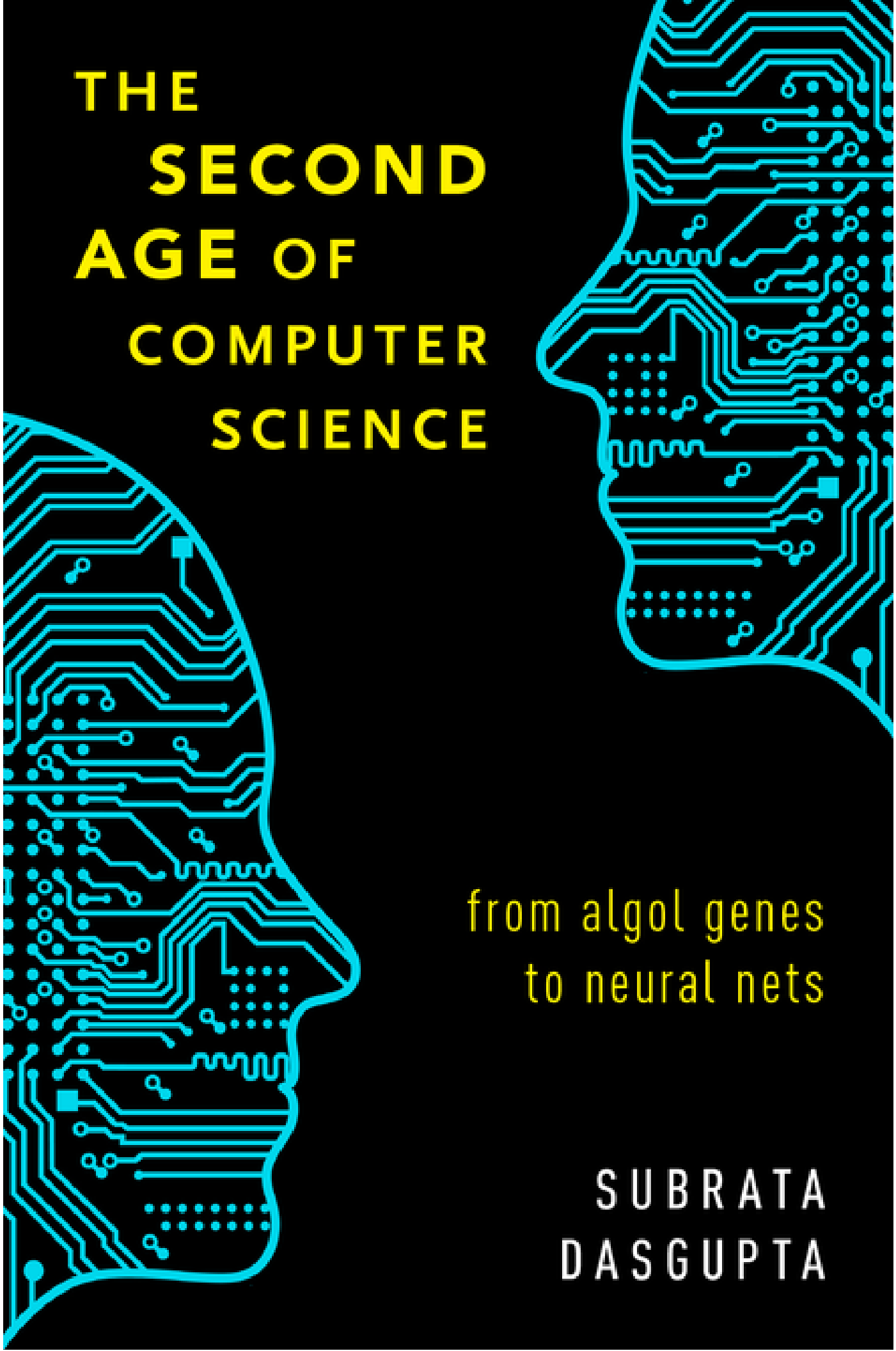
THE SECOND AGE OF COMPUTER SCIENCE



from algol genes to neural nets

SUBRATA DASGUPTA

THE SECOND AGE OF COMPUTER SCIENCE

The Second Age of Computer Science

FROM ALGOL GENES TO NEURAL NETS

Subrata Dasgupta

OXFORD
UNIVERSITY PRESS

OXFORD
UNIVERSITY PRESS

Oxford University Press is a department of the University of Oxford. It furthers the University's objective of excellence in research, scholarship, and education by publishing worldwide. Oxford is a registered trade mark of Oxford University Press in the UK and certain other countries.

Published in the United States of America by Oxford University Press
198 Madison Avenue, New York, NY 10016, United States of America.

Library of Congress Cataloging-in-Publication Data
Names: Dasgupta, Subrata, author.
Title: The second age of computer science : from ALGOL genes to neural nets / Subrata Dasgupta.
Description: New York, NY : Oxford University Press, 2018. |
Includes bibliographical references and index.
Identifiers: LCCN 2017057808 | ISBN 9780190843861
Subjects: LCSH: Computer science—History—20th century. | Genetic programming (Computer science) | ALGOL (Computer program language) | Neural networks (Computer science)
Classification: LCC QA76.17 .D363 2018 | DDC 004.09—dc23
LC record available at https://lccn.loc.gov/2017057808

9 8 7 6 5 4 3 2 1

Printed by Sheridan Books, Inc., United States of America

To

Sarmistha (Mithu) Dasgupta

and

Heidi Holmquist

Contents

Acknowledgments

THANK YOU JEREMY LEWIS, my publisher at Oxford University Press, who embraced this project from the day he received the manuscript. As always, it has been a pleasure working with him.

Thank you Anna Langley, editor at OUP, for all your help.

Lincy Priya steered this project very professionally from final manuscript to print. I thank her and her production team at Newgen.

My thanks to Victoria Danahy for her careful, nuanced, and informed copyediting. Her comments as "asides" were always interesting.

Thank you Elman Bashar who took time off from his own work to create the images that appear in this book.

Four anonymous reviewers read drafts of some of the chapters on behalf of the publisher and offered thoughtful and invaluable comments. I thank them, though I would have preferred to acknowledge them by name.

Finally, as always, my thanks to Mithu, Deep, Heidi, and Shome.

Prologue

P.1 THE FIRST AGE OF COMPUTER SCIENCE

In the 1960s a science the likes of which had never been quite known before came into being. At its epicenter were an idea and an artifact. The idea was *automatic computation*, the notion that symbol structures representing information about things in the world could be mechanically transformed into other symbol structures representing other pieces of information. The artifact was the *digital computer*, a machine for effecting such symbol or information processing. And with the naming of this science as *computer science*,[1] its teachers, researchers, and practitioners chose to call themselves *computer scientists*.

The genesis of computer science has a long and tenuous history reaching back to the first quarter of the 19th century, to pre–Victorian England, in fact.[2] Charles Babbage with whom the story of the birth of automatic computation properly began was a product of the Romantic Age in Europe.[3] By the time computer science assumed an identity and a name of its own in the 1960s, its gestation had spanned the long cultural period called the Modern Age and, indeed, when the science emerged, the world was in the first throes of the Postmodern era.

At any rate, as the 1960s drew to a close the discipline was in place. The first batches of formally trained computer scientists were emerging. Universities were offering bachelor's, master's, and doctoral degrees in the discipline. By the end of the decade, computer science could boast a certain level of maturity. It could claim the status of what philosopher of science Thomas Kuhn called in his seminal book *The Structure*

of Scientific Revolutions a scientific paradigm.[4] We may call this the *computational paradigm*.

A seemingly clear representation of this brand new, and quite unique, paradigm was offered in an influential document defining the contents of computer science circa 1968. *Curriculum 68* was a comprehensive report published under the auspices of the Association for Computing Machinery (ACM), founded in 1947 and the world's first professional society dedicated to computing. This document recommended a curriculum for academic programs in computer science, from "basic" through "intermediate" to "advanced" courses. The report was authored by a committee of 12 academics all drawn from North American universities—11 American and 1 Canadian—who had solicited and received input and opinions from over three-score consultants and reviewers.[5] It seems fair to say (despite its North American authorship) that publication of *Curriculum 68* implied a certain perceived maturity and stability of the computational paradigm as the '60s drew to a close. With historical hindsight let us call the state of the paradigm circa 1969, reflected in *Curriculum 68,* as marking the *first age of computer science*.

The subject matter of computer science, according to this report, consisted of three "major divisions." The first, *information structures and processes,* concerned the "representations and transformations of information structures and with theoretical models for such representations and transformations."[6] It included "data structures," "programming languages," and "models of computation."[7] The second division was named *information processing systems* and included such topics as "computer design and organization," "translators and interpreters," and "computer and operating systems."[8] The third division, called (somewhat confusingly) *methodologies*, consisted of "broad areas of application of computing which has common structures, processes and techniques" and included such topics as text processing, information retrieval, artificial intelligence (AI), and numerical mathematics.[9]

The message was clear: The basic stuff of the computational paradigm was *information*. Further support of this idea was given by the formation, in 1960, of the official world body on computing, called the International Federation for Information Processing (IFIP).

Unfortunately, the term "information" possesses two quite opposite meanings. Communication engineers use it to mean a commodity *devoid of meaning* that gets passed around in telephone and other electrical communication systems, rather as money is circulated in financial systems. The unit of meaningless information is the *bit*, allowing only two values, commonly denoted as "1" and "0." In contrast, in a commonsense or everyday connotation, information is an assertion about something in the world; here information has semantic content: It is *meaningful*. The only shared attribute of these oppositional notions of information is that information—meaningless or meaningful—reduces uncertainty.[10]

So which of these connotations was *Curriculum 68* referring to when they spoke of "information structures and processing" or "information processing systems"? The report does not explicitly clarify, but judging from the contents of the three major

"divisions" it would seem that what the authors had in mind was *predominantly* meaningful or semantic information. So we can say that as the 1960s came to an end the basic stuff of the computational paradigm was taken to be meaningful information, with the caveat that any piece of meaningful information is "ultimately" *reducible* to patterns of meaningless information, that is, patterns of 1's and 0's.

P.2 RECOGNIZING COMPUTER SCIENCE AS A SCIENCE OF THE ARTIFICIAL

The authors of *Curriculum 68* understood clearly that, despite its originality and strangeness, computer science was not an island of its own. Recognizing that certain branches of mathematics and engineering were "related" to computer science they added two other auxiliary divisions of knowledge, calling them *mathematical sciences* and *physical and engineering sciences*, respectively. The former included standard topics such as linear algebra, combinatorics, number theory, statistics and probability, and differential equations. The latter consisted almost entirely of topics in electrical and communication engineering.

But what *Curriculum 68* did *not* state is what *kind* of science computer science was conceived to be. How did it differ, for example, from such sciences as physics and chemistry or from the social sciences such as economics? What (if any) were their commonalities? What was the relationship between the brand new science of computing and the venerable field of mathematics? And what distinguished (if at all) computer science from the traditional engineering sciences?

These are metascientific questions. They pertain to the essential nature of a science (its ontology), its methods of inquiry (methodology), and the nature of the knowledge it elicits (its epistemology). Every paradigm governing a science includes metascientific propositions that collectively articulate the philosophical foundation of the science. If we take *Curriculum 68* as a representation of the computational paradigm at the end of the 1960s, then such a metascientific foundation was noticeably missing.

Yet the founding figures of computer science were very much cognizant of this aspect of their discipline. Largely in defense against skeptics who questioned whether computer science was at all a science, Allen Newell, Alan Perlis, and Herbert Simon, three of the founding faculty of the computer science department in the Carnegie Institute of Technology (later Carnegie-Mellon University), published in 1967 a short article in the journal *Science* titled "What is Computer Science?."[11] Their answer was, quite baldly, that computer science is the study of computers and their associated phenomena. The disbelievers charged that science deals with natural phenomena whereas the computer is an *artifact*, an instrument; and that instruments per se do not have their own sciences (for example, the science underlying the electron microscope is part of physics). In response, Newell and his coauthors maintained that though the computer is an instrument and an artifact, its complexity, richness, and uniqueness set it apart from other instruments, and thus the phenomena surrounding it cannot be adequately parceled out to an existing science.

Herbert Simon was a polymath who had already contributed seminally to administrative decision making, economics, psychology, philosophy of science, and, most pertinently, to the creation of AI, a branch of computer science.[12] Given his particular range of interests, it was almost inevitable that he would want to explore their shared scientific identity and how they differed from or were similar to natural science. The fruits of this exploration was his highly original book *The Sciences of the Artificial* (1969). Here, Simon dwelt on the nature of the artificial world and the things that populate it—artifacts. The artificial world, he pointed out, is to be distinguished from the natural world. The problem that especially intrigued him was how empirical propositions could at all be made about artificial phenomena and subject to corroboration or falsification in the sense that natural scientists construct and test empirical propositions about natural phenomena. In other words, he was arguing for the possibility of the artificial having its own distinct sciences. Thus computers, being artifacts, the empirical phenomena surrounding them belonged to the artificial world. If Simon's argument is pursued then we have an answer to our metascientific question: *computer science is a science of the artificial*.

P.3 THREE CLASSES OF COMPUTATIONAL ARTIFACTS

If we accept the authors and other contributors to *Curriculum 68* as representatives of the *first generation* of computer scientists, then this generation clearly recognized that the richness of the phenomena surrounding the computer lay in that the computer was not a singular, monolithic artifact; nor was it the only artifact of interest to computer scientists. Rather, there was an astonishing diversity of *computational artifacts*—artifacts that participate one way or another in information processing and automatic computation.

For instance, there was the Turing machine, a mathematical construction, embodying the *idea* of computing, invented in 1936 by the English mathematician and logician Alan Turing, at the time of King's College, Cambridge. Turing proposed that any process that "naturally" or "intuitively" we think of as computing can be realized by a version of the Turing machine.[13] This proposition was dubbed generously the "Turing thesis" by American logician Alonzo Church (1903–1995)—"generously" because Church had, about the same time, arrived at a similar proposition using a different kind of argument. And by the end of the 1960s, it was firmly ensconced as a central theoretical element of the computational paradigm. The Turing machine was undoubtedly a computational artifact but not of the kind most people think of when they use the term "artifact."

Of a more practical nature was yet another concept—the model of the stored-program digital computer conceived in the mid-1940s by a tiny group of pioneers and given voice by the Hungarian American mathematical genius John von Neumann; and thus often called (unjustly according to some), the "von Neumann computer model."[14] Here again is an invention of the human mind, thus an artifact.

The Turing machine and the stored-program computer model, both *abstract* entities, are just two of the varieties of computational artifacts that were represented in the computational paradigm as it was at the end of the '60s—and as reflected in *Curriculum 68*. In fact, one can place the varieties of computational artifacts into three broad classes.[15]

Material artifacts: These are governed by the laws of physics and chemistry. They occupy physical space, and their actions consume physical time. Physical computers and their constituent electronic circuits, input and output devices, and communication networks are the prime examples of material computational artifacts (known, more informally, as hardware).

Abstract artifacts: These exist as symbol structures, devoid of any materiality and thus not subject to physico-chemical laws. The Turing machine and the stored-program computer model are prime examples of this class, but there are others, such as programming languages and algorithms.

Liminal artifacts: These are inherently abstract yet their manipulation causes changes in material artifacts such as the transmission of electrical signals across communication paths, changing the physical states of circuits, activating mechanical devices, and so on. They also require a material infrastructure that enables their work to be done. Elsewhere, I have called such artifacts "liminal" because they straddle—are between and betwixt—the purely abstract and the purely material.[16] Computer programs ("software"), microprograms ("firmware"), and computer architectures are principal examples of liminal computational artifacts.

P.4 THE LIFE AND TIMES OF A PARADIGM

As Thomas Kuhn pointed out a scientific paradigm is a dynamic thing.[17] There are "holes" in its constituent bodies of knowledge that have to be plugged. Controversies must be resolved. Problems suggested by the overall paradigm must be solved. Solutions may have to be revised, even dismissed. The paradigm evolves and grows.

An artificial science is driven by the invention of artifacts. New artifacts stimulated by new social, cultural, technological, and economic needs come into being; and older artifacts, no longer of practical relevance, recede into the background or perhaps disappear altogether from scientific consciousness. The new artifacts give rise to new kinds of problems never recognized before. New questions are asked. The new problems engender new ways of thinking. Circumscribing the new artifacts, entirely new bodies of knowledge and philosophies crystallize: They form "subparadigms." Older bodies of knowledge may be discarded or forgotten, but they never disappear altogether, for one never knows how or when they might become relevant again. Most significantly, however, the fundamental structure of the paradigm does not change.

Kuhn controversially and somewhat disparagingly called this evolutionary growth of a paradigm "normal science" because he wanted to distinguish it from radical

challenges to an existing paradigm, so radical that they may cause the existing paradigm to be *overthrown* and replaced by a new one. Thus his terms "extraordinary science" and "paradigm shift"; thus his description of a scientific revolution. In the Kuhnian framework, normal science and paradigm shifts were the twin pillars of how a science progresses: the one incremental, constructive and *evolutionary*; the other iconoclastic, destructive or reconstructive and *revolutionary*.

P.5 THE SECOND AGE OF COMPUTER SCIENCE

Yet the history of an actual science is rarely as well structured as philosophers of science like to portray. Computer science may well have attained paradigmatic status by the end of the '60s. And there was, no doubt, Kuhnian-style normal science at work thereafter, adding bricks to the edifice created in the preceding decades, and enriching the paradigm as a whole. But computer science was still a precociously *young* science at the time. The computational paradigm was anything but stable. New, unanticipated forces, some technological, some economic, some social, some intellectual, impinged on it. The outcome was that entirely new kinds of problem domains arose over the next two decades that both significantly altered existing subparadigms and created new ones. In one or two situations, the core element of the computational paradigm—the von Neumann model of computing—would even be challenged.

Indeed, by the beginning of the 1990s, though there was not a paradigm *shift* in the Kuhnian sense—a revolution—the structure of the computational paradigm looked very different in many important respects from how it was at the end of the 1960s. So much so that it seems entirely reasonable to call the two decades from 1970 to 1990 the *second age of computer science*. My aim in this book is to narrate the story of this second age.

So what were the core themes of this new age?

P.6 THE QUEST FOR SCIENTIFICITY IN REAL-WORLD COMPUTING

First, this was a time when many computer scientists became seriously reflexive about their discipline—not so much whether computer science was a genuine science but about the nature of the science itself. Herbert Simon's notion of the sciences of the artificial mentioned in Section P.2 obviously pointed computer scientists in a certain direction. But outside the AI community—Simon being a seminal figure in AI—it is not clear whether *The Sciences of the Artificial* had much of an *immediate* impact on other computer scientists.

The intellectual origins of computer science lay in logic and mathematics. As the 1960s ended the presence of these venerable disciplines was still very much visible in the shape of two branches of computer science. One was numerical analysis, concerned with the approximate numerical solutions of continuous mathematical problems—which was what began the quest for automatic computation in the first place and that particularly flourished in its first age of computer science.[18] The other

was automata theory concerned with the study of highly abstract, mathematical models of computation.

For some computer scientists, automata theory *was* the "science" in "computer science." This suggested, indeed assumed, that computer science was a *mathematical* science. Research papers, monographs, and textbooks devoted to automata theory were full of axioms, definitions, theorems, corollaries, and proofs in the traditions of mathematics and mathematical logic. The problem was that not all computer scientists were convinced that automata theory had any direct influence on, or relevance in, the design, implementation, and understanding of *real* computational artifacts. As Maurice Wilkes, one of the pioneers of automatic computing, put it in an informal talk delivered in 1997, one did not need to know automata theory to build physical computers, algorithms, or software. This can be contrasted, he remarked, to the relationship between Maxwell's equations in electromagnetic theory and radio engineering. If one wanted to design a radio antenna, Wilkes said, one had better know something about Maxwell's equations. Automata theory had no such linkages with much of the rest of computer science (including numerical analysis).[19]

Moreover, there were substantial parts of the computational paradigm that at the end of the '60s did not seem "scientific" at all. For example, a substantial part of computer architecture, the branch of computer science concerned with the functional structure and design of computers, was essentially a heuristic, craft-like discipline having little theoretical content at all.

So also in the realm of programming: Even though there was recognition that this entailed a whole new kind of artifact (liminal in the sense I have used here), deserving a name of its own—"software," coined in the mid-1960s—there was a perceptible sense of inadequacy about the way programs and software were being designed and implemented. In 1968, the term "software engineering" was invented to indicate that software had evolved to a level of complexity that demanded the same kind of intellectual respect that was accorded to the artifacts civil, mechanical, and other contemporary branches of engineering dealt with. Yet it was far from clear what shape this respect should take.

Programming languages—abstract artifacts—had reached a more scientifically respectable state by the beginning of the second age. In particular, the '60s decade was rich in its advancement of a theory of the syntax of programming languages, concerned with the rules governing grammatically correct forms of programs expressed in programming languages. But *semantic* considerations—what the elements of a programming language meant and how to characterize their meanings, how language implementers could unambiguously and correctly translate programs in a given language into machine-executable code, how programmers could objectively interpret the meanings of language elements in the course of writing programs—were a different matter. Theories of the semantics of programming had indeed emerged in the late '60s, but very little of these theories had been adopted by the designers and implementers of "real-world" languages and industrial-grade software. Instead, these designers and implementers resorted to natural language and informality in defining the semantics of their programming languages.

Collectively, these concerns became audible as sounds of discontent and dissent. They represented a desire for a *science of real-world computing* that would play the same role in the design, implementation, and understanding of real, utilitarian computational artifacts as, say, the engineering sciences of strength of materials and fluid mechanics play, respectively, in structural and aeronautical engineering.[20]

As we will see, this quest for *scientificity*—and how this quest elicited responses—constituted one of the most significant markers of the second age of computer science.

P.7 COMPUTING AS A HUMAN EXPERIENCE

This was also a time when some computer scientists developed a heightened consciousness about computing *as a human experience*. This too did not arise de novo. The makers of the first age of computer science had clearly recognized that although automatic computation with minimal human intervention was their fundamental mission, humans were not thereby absolved of all responsibilities. Having built physical computers they could not, in the manner of Pontius Pilate, wash their collective hands and look the other way. After all, at the very least humans had to communicate their intentions to machines. The very first book on programming, authored by Maurice Wilkes, David Wheeler, and Stanley Gill of Cambridge University, was called, significantly, *Preparation of Programmes for an Electronic Digital Computer* (1951). (In Britain, the English spelling "programme" still prevailed to signify this artifact rather than the American spelling that would eventually be generally adopted.)[21] Computer programs, reflecting the users' computational intentions, had to be *prepared* by human beings. The invention and design of the first programming languages, reaching back to German engineer Konrad Zuse's *Plankalkül* in 1945[22] through David Wheeler's invention of an assembly language for the Cambridge EDSAC in 1949[23] to the development by the American John Backus and his collaborators in IBM of FORTRAN[24] in 1957 were clear manifestations of the computer pioneers' awareness of incorporating humans into the computing experience. The remarkable transatlantic collaboration leading to the design of the Algol programming language between 1958 and 1963[25] offered further evidence of this awareness.

But with the emergence and development of the computational paradigm in the late '50s and especially through the '60s, two quite different and contrasting manifestations of the human–computer connection emerged. In both cases it was not the incorporation of humans into the computing milieu but rather the opposite: incorporation of computing into the human experience.

P.8 PROGRAMMING AS AN INTELLECTUAL ACTIVITY

Along one front, in the late 1960s, the Dutch mathematician-turned-"humble programmer" (as he would style himself[26]) Edsger Dijkstra articulated a vision of programming as, fundamentally, a human activity.[27] Dijkstra was among the very first advocates of a new *mentality*: treating programming not only, or not necessarily, as a

means for communicating tasks to computers but as an *intellectual activity in its own right* regardless of the presence or absence of physical computers.

This mentality can be viewed as a provocative *challenge* to one of the key ontological assumptions underscoring the prevailing computational paradigm in the first age: that it should concern itself with computing as a *mechanical* information process—an assumption that, of course, was rooted in Turing's vision of computing. We will see that although this new human-centered view of programming was rooted in the 1960s it engendered an intellectual *movement* in the next two decades called "structured programming." Human-centeredness formed the second marker of a new age of computer science.

P.9 TOWARD A UNIFIED SCIENCE OF INTELLIGENCE

Neither Dijkstra nor the other makers of the structured programming movement extended their idea of programming as a human activity into the realm of human *cognition* itself. Envisioning programming as an intellectual activity did not imply that the mind itself was a computational object—a *natural* computational object, not an artifact. The idea of the mind as a machine has its own incredibly long history reaching back to antiquity.[28] The modern notion of the mind as a computational machine, however, originated in the 1940s, coincident with the time that digital computing was "in the air." For instance, influenced by *analog* computing machines such as the differential analyzer (originally built by American engineer Vannevar Bush in 1931),[29] the British psychologist Kenneth Craik suggested in 1943 that thinking was a kind of computational process.[30] This daring hypothesis was given much more concrete form in 1958 by Allen Newell, Cliff Shaw, and Herbert Simon. In a paper published in an influential psychology journal they suggested just how the newly emergent computational paradigm offered a promising, indeed powerful, model of human problem solving.[31] They backed this model by a working computer program called Logic Theorist, implemented in 1956, which could prove theorems in mathematical logic.[32] In effect, this paper was seminal in the emergence in the 1960s of AI as a subparadigm of the computational paradigm that would connect computers with human cognition; and it would persuade psychologists, linguists, anthropologists, and philosophers to join with AI researchers to create an interdisciplinary science of mind they called *cognitive science*.

AI's agenda in this regard was not *exactly* that of cognitive science: The latter's allegiance was to human (and to a lesser extent, other animal) cognition; computation was a means to that end. AI's agenda, at its most ambitious, was to construct a unified science of intelligence that would embrace *both* artifacts and humans. And, in the 1970s and 1980s, this project evolved in a number of surprising directions.

One was the consolidation of the idea that symbols constituted the fundamental "stuff" of thought in both machines and mind. So a unified science of intelligence would be centered on *symbol processing*.

Along a second path lay what we may call *computational epistemology*. Epistemology—the study of the nature of knowledge—was of course one of the great pillars of the philosophy of mind, having origins reaching back to both Western and Eastern antiquity. But what AI ushered in were new concepts about knowledge, in particular, how knowledge is represented in cognitive systems, both natural and artificial, and how knowledge could be processed in the course of thinking. This new computational epistemology was another major marker of the second age of computer science.

But even as symbolism was being consolidated, even as a symbol-based epistemology was being vigorously explored, the very idea of symbols as the stuff of intelligence and cognition would be challenged. Stimulated by the heady excitement of parallel processing (see the next section) and by the findings of neurobiology, neurologically inspired models of cognition would begin to seriously challenge the symbolist subparadigm governing AI. This was yet another core signifier of the second age of computer science.

P.10 LIBERATING COMPUTING FROM THE SHACKLES OF SEQUENTIALITY

The evolution of electronic computing from its very inception in the early 1940s had always been shaped by the *technological imperative*: the force of the technology with which material computational artifacts—memories, processors, peripheral devices, communication paths—were built. But as the '60s gave way to the '70s this imperative assumed a particularly dramatic form: the phenomenal progress in integrated-circuit (IC) technology resulting in the fabrication of IC chips (the units of production) of sharply increasing density of components and (thus) computing power, and the concomitant decrease in unit cost of circuit components. By the early 1970s the term "large-scale integration" was part of the vocabulary of hardware designers; it had also infiltrated the consciousness of computer scientists. By the end of the decade the new operative term on everyone's lips was "very large-scale integration."

The impact of this rapid evolution on IC technology—some would claim it was a revolution—was manifold. For computer scientists it made practical what had been more a promise in the 1960s: liberating computing from the shackles of sequentiality. Thus the second age of computer science was marked by the exuberant reality of *parallel computing*—inside the computer, between computers, within algorithms, within programs, in programming languages. This prospect had certainly emerged in the first age, but it was in the second age that computer scientists seriously came to *think* parallelism.

P.11 SUBVERTING THE VON NEUMANN PRINCIPLE

A very different phenomenon arose in the '70s and '80s: an act of serious rebellion, in fact. There were some people who wished to subvert the very core of the prevailing

computational paradigm—the so-called von Neumann principle. They wished to challenge the von Neumann-style *architecture of computing* as had been adopted since the birth of modern computing. These rebels wanted to be revolutionaries: They desired to bring about a veritable *paradigm shift* in computer science. A part of this desire originated in the annoyance with certain "bottlenecks" inherent in von Neumann architecture, in part in the impatience with sequential computing, in part in the excitement engendered by the architecture of the brain, in part in the perceived imperfections of conventional programming languages, and in part in the advent of very large-scale integration technology.

This subversive mentality and its consequences formed the fifth marker of a distinctive second age of computer science.

P.12 CAVEATS FOR THE READER

So the principal themes that characterize what I am calling the *second age of computer science,* circa 1970–1990 are (a) the quest for scientificity in real-world computing; (b) computing as a human experience; (c) the search for a unified theory of intelligence encompassing both machine and mind; (d) liberation from the shackles of sequentiality; and (e) subverting the von Neumann core of the computational paradigm. These themes appear and reappear in various guises in the seven chapters that follow.

But some caveats are in order.

One. This book is intended to continue the story of computer science beyond where my earlier book *It Began with Babbage* (2014) ended. That book had chronicled the genesis of computer science and had ended circa 1969 by which time (as noted earlier) an academic discipline of computer science and the central features of the computational paradigm had been established. *Curriculum 68* was a manifesto of what I call here the first age of computer science.

The present story covers roughly the '70s and '80s, the two decades forming what I am calling the second age of computer science. But such words as "circa" and "roughly" are important here, especially concerning the beginning of this second age. As the reader will see, this beginning often overlaps the waning years of the first age. Concepts and ideas from the latter appear like the painter's pentimento in this story. Likewise, ending the story at about 1990 does not imply that this age ended abruptly. Rather, circa 1990 was when a pair of remarkable new computational artifacts made their appearances, ones that would have vast social and commercial consequences but also give new shape to computer science itself. These artifacts were the Internet (whose own history began in the 1960s, in fact), an American invention, and the "World Wide Web" invented by the Englishman Tim Berners-Lee. The third age of computer science would begin. Thus it is more appropriate to say that circa 1990 was a marker of the *waning* of the second age.

Two. The chapter sequence in this book is not a narrative that maps onto the sequential passage of historical time. Rather, the seven chapters are seven overlapping

"spaces," like adjacent geographic regions, *within* each of which creative events occur in roughly sequential time, but *between* which events are or may be concurrent. The history I have narrated here is thus really seven histories unfolding concurrently over the same historical time period. We may well call this an instance of *parallel history*. But they are not mutually independent. Like all complex and interesting cases of parallel processing, there are interactions, linkages, and cross-connections among these parallel histories. So the reader will find frequent cross references between chapters.

P.13 WHAT KIND OF HISTORY IS THIS?

Historians tend to identify themselves and their writings (*historiography)* with different *genres* of history. So what genre does this book fall under?

As the reader will see, this is a history of creativity, imagination, the origin, and evolution of ideas and knowledge in the realm of computer science; it is also a history of how computer scientists perceived, identified, invented, and/or discovered computational themes and problems, and how they reasoned and critically responded to such problems. Thus this book is primarily a blend of *intellectual history* and *cognitive history*. The former is of older vintage, the latter is of very recent origins.

Intellectual history itself has a history. Previously (under the name of *history of ideas*) it was concerned with tracing how ideas and concepts evolve, grow, and spawn other ideas over historical time.[33] More recently intellectual historians have turned their attention to the study and interpretation of historical texts and the contexts and (especially) the language in which they are produced.[34] It so happens that both the "old" and the "new" versions of intellectual history have places in this book, though more notably as history of ideas.

Cognitive history constructs the history of some creative realms (such as science, art, technology, or religion) by delving into the cognitive attributes of the creative individuals involved. Cognitive-historical explanations of acts of creativity are grounded in such cognitive issues as the goals, needs, or desires wherein ideas originate, the creative being's system of beliefs, knowledge, modes and style of reasoning and critical thinking.[35]

To repeat, the present work lies at the juncture of intellectual history and cognitive history.

But, like any other human endeavor, computer science has its *social* and *cultural* dimensions. For instance, the fruits of a science must be *communicated* or disseminated. Journals are created, societies are formed, conferences are held. Networks of relations are formed. Computer science is no exception to this. Most of the major computing societies—in the United States, Canada, the United Kingdom, Australia, Germany, and India—were established in the first age of computer science including, most notably, IFIP, a kind of United Nations of computing. The second age saw the expansion of this communicative–disseminative aspect of computer science in the form of new

societies, journals, and conferences, and the formation of "special-interest groups," "technical committees," and "working groups."

Another sociocultural aspect pertains to gender. In *It Began with Babbage* I noted that the genesis of computer science was almost exclusively the province of men. There were a very small number of women who contributed importantly to the early development of electronic computing, most famously Grace Murray Hopper and Jean Sammet, but the others have sadly and unjustifiably remained largely unknown until very recent times.[36] In the second age women still remained very much a minority. Yet a small but highly visible coterie made their presence felt through the pages of computer science journals, in universities and college academic departments, and at conferences.

Yet another sociocultural feature of the first age was that the people who created computer science were, without exception, from the West.[37] By the end of the second age this demographic would change markedly, some might even say dramatically. Publications would bear names of very definitely non-Western character. The computer science community became increasingly and noticeably multiethnic and multicultural.

Finally, as we will see, computer science as a scientific discipline is practiced both in academia and in the corporate world. And as is well known there are distinct cultural differences between these two societies. Thus the following interesting questions arise: How was the development of computer science *distributed* across these two cultures? Did the two cultures impose their imprint on the nature of their respective activities in computer science? And if so, in what manner?

Some of these sociocultural elements of the second age will appear in this narrative—but only implicitly, insofar as they relate to the intellectual and cognitive story being told. The *sociocultural history* of computer science addressing explicitly such issues as gender, race, ethnicity, and the "two cultures" needs to be written. But this is not that book.

NOTES

1. Or some variant or other-language equivalent thereof. For example, "computing science" or "computational science" in English-speaking lands, *informatik* or *informatique* in European countries.

2. For an account of this history see S. Dasgupta, 2014. *It Began with Babbage: The Genesis of Computer Science.* New York: Oxford University Press.

3. See N. Roe (ed.), 2005. *Romanticism*. Oxford: Oxford University Press.

4. T. S. Kuhn, [1962] 2012. *The Structure of Scientific Revolutions* (4th ed.). Chicago: University of Chicago Press.

5. ACM Curriculum Committee, 1968. "Curriculum 68," *Communications of the ACM*, 11, 3, pp. 151–197.

6. ACM Curriculum Committee, 1968, p. 156.

7. Ibid.

8. Ibid., p. 155.

9. Ibid.

10. P. S. Rosenbloom, 2010. *On Computing: The Fourth Great Scientific Domain*. Cambridge, MA: MIT Press.

11. A. Newell, A. J. Perlis, and H. A. Simon, 1967. "What is Computer Science?," *Science*, 157, pp. 1373–1374.

12. For a detailed examination of Simon's polymathic mind and work see S. Dasgupta, 2003. "Multidisciplinary Creativity: The Case of Herbert A. Simon," *Cognitive Science*, 27, pp. 683–707.

13. A. M. Turing, 1936. "On Computable Numbers with an Application to the *Entscheidungsproblem*," *Proceedings of the London Mathematical Society,* 2, pp. 230–236.

14. The history of the origins of the von Neumann model is complex and somewhat controversial. See Dasgupta, 2014, pp. 108–133.

15. Ibid., p. 4.

16. Ibid.

17. Kuhn, [1962] 2012.

18. Dasgupta, 2014, pp. 10 *et seq,* p. 279.

19. Cited by P. J. Bentley, OUPblog, Oxford University Press, June 18, 2012: www.http//blog.oup.com/2012/06/maurice-wilk. Retrieved February 12, 2014. Wilkes had, in fact, expressed this view in conversations with the present author.

20. S. P. Timoshenko, [1953] 1983. *History of Strength of Materials*. New York: Dover; W. G. Vincenti, 1990. *What Engineers Know and How They Know It*. Baltimore, MD: Johns Hopkins University Press.

21. M. V. Wilkes, D. J. Wheeler, and S. Gill, 1951. *Preparation of Programmes for an Electronic Digital Computer*. Cambridge, MA: Addison-Wesley.

22. F. L. Bauer and H. Wössner, 1972. "The 'Plankalkül' of Konrad Zuse: A Forerunner of Today's Programming Languages," *Communications of the ACM*, 15, pp. 678–685.

23. D. J. Wheeler, 1949. "Automatic Computing with the EDSAC," PhD dissertation, University of Cambridge.

24. J. W. Backus, R. W. Beeber, S. Best, R. Goldberg, L. M. Halbit, H. C. Herrick, R. A. Nelson, D. Sayre, P. B. Sheridan, H. Stern, I. Ziller, R. A. Hughes, and R. Nutt, 1957. "The FORTRAN Automatic Coding System," *Proceedings of the Western Joint Computer Conference,* Los Angeles, CA, pp. 188–197.

25. P. Naur (ed.), et al. (1962–1963). "Revised Report on the Algorithmic Language ALGOL 60," *Numerische Mathematik*, 4, pp. 420–453.

26. E. W. Dijkstra, 1972. "The Humble Programmer," *Communications of the ACM,* 15, 10, pp. 859–866.

27. E. W. Dijkstra, 1965. "Programming Considered as a Human Activity," pp. 213–217 in *Proceedings of the 1965 IFIP Congress*. Amsterdam: North-Holland.

28. M. A. Boden, 2006. *Mind as Machine* (two volumes). Oxford: Clarendon.

29. Dasgupta, 2014, p. 92.

30. K. J. W. Craik, [1943] 1967. *The Nature of Explanation*. Cambridge: Cambridge University Press.

31. A. Newell, C. J. Shaw, and H. A. Simon, 1958. "Elements of a Theory Human Problem Solving," *Psychological Review,* 65, pp. 151–166. See also Dasgupta, 2014, p. 233.

32. A. Newell and H. A. Simon, 1956. "The Logic Theory Machine: A Complex Information Processing System," *IRE Transaction on Informaton Theory,* IT-2, pp. 61–79. See also Dasgupta, 2014, pp. 229–232.