Philosophy of Computer Science: An Introductory Course

WILLIAM J. RAPAPORT
State University of New York at Buffalo

Abstract: There are many branches of philosophy called "the philosophy of *X*," where *X* = disciplines ranging from history to physics. The philosophy of artificial intelligence has a long history, and there are many courses and texts with that title. Surprisingly, the philosophy of computer science is not nearly as well-developed. This article proposes topics that might constitute the philosophy of computer science and describes a course covering those topics, along with suggested readings and assignments.

During the Spring 2004 semester, I created and taught a course on the Philosophy of Computer Science. The course was both dual-listed at the upper-level undergraduate and first-year graduate levels and cross-listed in the Department of Computer Science and Engineering (CSE) (where I am an associate professor) and the Department of Philosophy (where I have a courtesy appointment as an adjunct professor) at State University of New York at Buffalo ("UB").

The philosophy of computer science is not the philosophy of artificial intelligence (AI); it includes the philosophy of AI, of course, but extends far beyond it in scope. There seem to be less than a handful of such broader courses that have been taught: A Web search turned up some three or four that were similar to my course in both title and content.[1] There are several more courses with that title, but their content is more accurately described as covering the philosophy of AI. The philosophy of computer science deserves more exposure at the university level. The UB course was popular (with an enrollment of just under fifty), and the students found it valuable, not only for its coverage of topics in the philosophy of computer science, but also for the critical-thinking skills they learned (see p. 322). This article presents my ideas on what a course in the philosophy of computer science might look like.

 0145-5788 pp. 319–341

Why teach the philosophy of computer science? And why teach it in a computer science department rather than in a philosophy department? As a professor of computer science with a Ph.D. in philosophy (and a previous career as a philosophy professor), I've long been interested in philosophical issues in computer science in general and artificial intelligence in particular. My colleague Stuart C. Shapiro in the UB CSE department had urged me to develop some philosophy courses for our students. Initially, I had resisted this, not being sure that such courses would be acceptable to my department or—more importantly—taken by enough students. Moreover, my colleague Randall R. Dipert in our philosophy department regularly offered an undergraduate course in the philosophy of AI, with which I didn't want to compete.

However, there were many metaphysical, epistemological, and ethical issues that I thought were of interest in the non-AI part of computer science, many of which have only recently begun to be examined in detail by philosophers and philosophically-oriented computer scientists, and many of which shed new light on classical topics in philosophy. This article surveys them and offers some interesting readings that deserve to be better known. Moreover, a course such as this can serve as an introduction to philosophy for computer science students, an introduction to issues in computer science for philosophy students, a capstone course for senior undergraduate computer science students, or perhaps an overview course for beginning computer-science graduate students.

Syllabus

The course syllabus was organized around a set of questions whose various answers we examined during the semester:[2]

1. **What is philosophy?** In particular, what is "the philosophy of X" (where X = things like science, psychology, history, etc.)? (These questions are especially important to discuss in a course primarily aimed at computer science students, who might have misleading ideas of what philosophy is all about—or no idea at all.)

2. **What is computer science?** (Although the "final" answer to this question may simply be the extensional "whatever computer scientists do," this is a reasonable issue to discuss, even if there is no intensional answer. The following subquestions indicate some of the interesting issues that this main question raises.) (a) What is science? What is engineering? (b) Is computer science a science, or is it a branch of engineering? (c) If it is a science, what is it a science of? (d) Is it a science of computers (as some authors say)? (e) What, then, is a computer? (f) Or is computer science a science of computation (as other authors say)? (g) What, then, is computation? (h) What is an algorithm?

Is an algorithm different from a procedure? Many authors say that an algorithm is (like) a recipe; is it, or are there important differences? (i) What are Church's and Turing's "theses"? (j) Some authors claim that there are forms of computation—often lumped together under the rubric "hypercomputation"—that, in some sense, go "beyond" Turing-machine (TM) computation: What is hypercomputation"?

3. **What is a computer program?** (a) What is the relation of a program to that which it models or simulates? What is simulation? (b) Are programs (scientific) theories? (c) What is an implementation? (d) What is software? How does it relate to hardware? (e) Can (or should) computer programs be copyrighted, or patented? (f) Can computer programs be verified?

4. **What is the philosophy of artificial intelligence?** (a) What is AI? (b) What is the relation of computation to cognition? (c) Can computers think? (d) What are the Turing Test and the Chinese Room Argument?

5. **What is computer ethics?** (This, like the philosophy of AI, is a vast question, deserving of its own course and having many textbooks devoted solely to it. For my purposes, I decided to focus on questions that don't seem to be the typical ones asked in such a course.) (a) Should we trust decisions made by computers? (b) Should we build "intelligent" computers?

The remainder of this paper surveys these topics, suggests readings, discusses the sorts of assignments I gave, and presents some student reactions.[3]

Textbooks

Unfortunately, there is no textbook that exactly covers the above topics. Three possibilities were offered to the students as recommended texts: Luciano Floridi's *Philosophy and Computing* (1999), Timothy Colburn's *Philosophy and Computer Science* (2000), and Floridi's *Blackwell Guide to the Philosophy of Computing and Information* (2004). The first two are monographs offering the authors' points of view. There is nothing wrong with this, of course, but I preferred a more neutral approach for the sort of course that I had in mind. Moreover, the topics covered in each of these had a relatively small intersection with my topics. The third book is an anthology, but—again—there was only a small overlap with my topics, and, in any case, I preferred that my students read primary sources rather than overviews.

There are other sources, of course: A special issue of the philosophy journal *The Monist* (82:1 [1999]) was devoted to the philosophy of computer science. The journal *Minds and Machines: Journal for Artificial Intelligence, Philosophy, and Cognitive Science* is almost

entirely devoted to philosophy of computer science, broadly construed. And about half of the articles in the *Journal of Experimental and Theoretical Artificial Intelligence* are on philosophy of computer science. Finally, an excellent website, "Computational Philosophy," is moderated by John Taylor (http: //www.crumpled.com/cp/).[4] In the sections that follow—and more extensively on the course website (see note 3)—I recommend appropriate readings for the topics that we covered.

Topics and Readings

What is philosophy? A typical advanced philosophy course in a philosophy department normally does not need to address the question of what philosophy is, but I felt that a course whose principal audience was computer-science students needed to. I suspect that many such students feel that philosophy is a "soft" subject where there are no answers, so everyone's opinion is equally good.[5] In contrast, I hoped to present to the students a view of philosophy as an analytical and critical discipline that could be of value to them.[6]

I began with a brief history of western philosophy, beginning with Socrates' and Plato's view of the philosopher as "gadfly," challenging others' assumptions. I offered my own definition of philosophy as the search for truth in any field by rational means (which might be limited to deductive logic, or might be extended to include empirical scientific investigation). And we defined the "philosophy of X" as the study of the fundamental assumptions and main goals of any discipline X.

I briefly covered some of the basic principles of critical thinking and informal argument analysis, including the following notions:

1. "argument" (a set of premises and a conclusion)
2. "premise" (a Boolean proposition used to support a conclusion)
3. "conclusion" (a Boolean proposition that someone tries to convince you of by means of a logical argument)
4. "valid argument" (an argument is valid iff it is impossible for the premises all to be true yet for the conclusion to be false; this semantic notion can also be supplemented with a syntactic one: an argument is (syntactically) valid iff it has the form of any of a given standard set of argument forms that are (semantically) valid, such as Modus Ponens)
5. "factual argument" (this is a non-standard, but useful, notion:[7] an argument is factual iff all of its premises are true)
6. "sound" (an argument is sound iff it is factual and valid).[8]

I will have more to say about this when I discuss the course assignments, but I should point out that Computing Curricula 2001's "Social and Professional Issues" knowledge area includes the item "Methods and Tools

of Analysis" (SP3), which covers precisely these sorts of argument-analysis techniques (http://www.computer.org/education/cc2001/final/sp.htm#SP-MethodsAndTools).

As a reading assignment, I asked the students to read at least one of a variety of brief introductions to philosophy (*e.g.*, Plato's *Apology*; Colburn 2000: chaps. 3–4; Audi 2001), and I listed Mark B. Woodhouse's *Preface to Philosophy* (2003) as another recommended textbook for the course.

What is computer science? We began the first major section of the course by discussing the reasons one might have for asking what a discipline is: There are, of course, philosophical—primarily ontological—reasons. But there are also political reasons, especially in the case of a discipline such as computer science, which can be found both in (arts-and-)science faculties as well as in engineering faculties (sometimes in both at the same institution!), or even in its own faculty (either accompanied by other departments in, say, an informatics faculty or else by itself). Then, too, there is the question of the relationship between computer *science* and computer *engineering*.

We surveyed the following answers that have been given to the question "What is computer science?"

- It is a *science of computers and surrounding phenomena* (such as algorithms, etc.) (Newell, Perlis, and Simon 1967).
- It is the *study* (*N.B.* not "science") *of algorithms and surrounding phenomena* (such as the computers they run on, etc.) (Knuth 1974).
- It is the empirical study ("*artificial* science") of the phenomena surrounding computers (Newell and Simon 1976; cf. Simon 1996).
- It *is* a *natural* science, not of computers or algorithms, but of *procedures* (Shapiro 2001).
- It is *not* a science, but a branch of *engineering* (Brooks 1996).
- It is the body of knowledge dealing with information-transforming *processes* (Denning 1985).
- It is the study of *information* itself (Hartmanis and Lin 1992).

Note that several of these (especially the first two) might be "extensionally equivalent" but approach the question from very different perspectives: Some emphasize the computer (hardware); others emphasize algorithms, processes, procedures, etc. (software), or even something more abstract (*e.g.*, information). An orthogonal dimension focuses on whether computer science is a science or perhaps something else (a "study," a "body of knowledge," an engineering discipline, etc.). And, of course, the name itself varies (computer science, computing science, informatics, etc.), often for political, not philosophical, reasons.[9]

Is "computer science" science or engineering? The question of whether computer science is really a science or else is really a branch of engineering has been the subject of several essays. It has had special relevance at UB ever since our former Department of Computer Science, housed in the Faculty of Natural Sciences and Mathematics, merged with several computer engineers[10] from our former Department of Electrical and Computer Engineering to form a new Department of Computer Science and Engineering housed in the School of Engineering and Applied Sciences. This is not only confusing to read about, but has given rise to a certain identity crisis for both our students and faculty, and I thought it would provide interesting local color to an investigation of the nature of science and of engineering.

We first turned to the question "What is science?" discussing both its goals (should it merely *describe* the world—as Ernst Mach thought [cf. Alexander 1967: 118–119]—or *explain* it?) as well as the nature of its theories (are they merely instrumentalist, or realist?). We looked at debates over scientific method (is it experimental and cumulative, or does it proceed by paradigm and revolution?) and its branches (is mathematics a science?). The primary readings on science were selections from David Papineau's "Philosophy of Science" (1996) and chapters from John G. Kemeny's *A Philosopher Looks at Science* (1959).

We next looked at the history of engineering (Michael Davis's *Thinking Like an Engineer* [1998] is especially useful), discussing engineering as applied science, as defined in terms of professional education, and as a design activity (Petroski 2003). And we looked at a definition of computer science as a new kind of engineering that studies the theory, design, analysis, and implementation of information-processing algorithms (Loui 1987, 1995).

What is a computer?—Part I. Insofar as computer science is the science (or study) primarily of computers, the next reasonable question is: What is a computer? This is a large topic, and I divided it into two parts.

The first part was a survey of the history of computers. I presented this in terms of two parallel goals: the goal of building a computing machine, and the goal of providing a foundation for mathematics. As I see it, these were two more-or-less independent goals that converged in the first half of the twentieth century. (Whether or not this is a historically accurate way of looking at the matter is itself an interesting question; in any case, it is certainly a convenient way to organize the topic pedagogically.) Our discussion of the first goal involved the contributions of Babbage, Aiken, Atanasoff and Berry, Turing, and Eckert and Mauchly. The contributions of Leibniz, Boole, Frege, Hilbert, Turing, Church, and Gödel made up the overview of the second goal.

The history of computers is a large topic, and we did not spend much time on it. Consequently, the assigned readings were intended only to give the students a flavor of the main events. I prepared a website, "A Very Brief History of Computers,"[11] based on the IEEE's "Timeline of Computing History"[12] and containing links for further information, and I asked the students to read O'Connor and Robertson 1998 (on Babbage), Simon and Newell 1958 (pp. 1–3 are also on Babbage), and Ensmenger 2004 (on the controversy over who deserved the US patent for the first computer).

What is an algorithm?—Part I. The other main answer to the question of what computer science studies is: algorithms. So, what is an algorithm? We began our two-part investigation of this by first considering what computation is. One informal, introductory-computer-science-style explanation proceeds as follows: A function f (viewed as a set of ordered pairs, or "inputs" and "outputs") *is computable* means by definition that there is an "algorithm" that computes f, *i.e.*, there is an algorithm A such that for all input i, $A(i)=f(i)$, and A specifies how f's inputs and outputs are related (or how f's outputs are produced by its inputs). Then an *algorithm for a problem P* can be characterized as a finite procedure (*i.e.*, a finite set of instructions) for solving P that is:

1. unambiguous for the computer or human who will execute it; *i.e.*, all steps of the procedure must be clear and well-defined for the executor, and
2. effective; *i.e.*, it must eventually halt, and it must output a correct solution to P.[13]

It became an interesting exercise as we went through the semester to compare the different (informal) explications of 'algorithm,' no two of which seem to be equivalent. This makes Turing's accomplishment all the more interesting!

With this informal exposition in mind, we then turned to a careful reading of Turing's *magnum opus*, "On Computable Numbers" (1936). There are several versions on the Web, though the most trustworthy is the one reprinted in Davis 1965. When I say "careful reading," I mean it: We spent an entire eighty-minute class doing a slow, "active," line-by-line reading of as much of it as we could.[14] I strongly recommend that all computer-science students (as well as computationally-oriented philosophy students, of course) do this at least once in their lives. In addition to being one of the most significant scientific papers of the twentieth century, it is also fascinating, well-written, and contains many interesting philosophical insights. The students told me afterward that this slow reading was one of the highlights of the course.

We also discussed the history of the mathematical investigation of the concept "computable," and discussed the relationship of (1) Turing's thesis that a function is (informally) computable if and only if it is TM-computable to (2) Church's thesis that a function is (informally) computable if and only if it is lambda-definable (which is logically equivalent to being recursive and, of course, to being TM-computable).

Besides Turing 1936, I also asked the students to read Leon Henkin's "Are Logic and Mathematics Identical?" (1962), which has a good discussion of the history of logic and the foundations of mathematics that led up to Turing's analysis, and Gabor Herman's "Theory of Algorithms" (1983), which discusses the informal notions of "algorithm" and "effective computability" and provides a good background for Turing 1936. I also especially recommend (to instructors, if not to students) Robert I. Soare's "Computability and Recursion" (1996) for the clarity it brings to the history of the competing analyses of 'computable' (*e.g.*, how Turing's Thesis differs from Church's Thesis).

What is a computer?—Part II. Armed with this background, we turned to philosophical questions surrounding the nature of computers. John Searle's "Is the Brain a Digital Computer?" (1990) argues that *everything* is a digital computer (which seems to trivialize the question), and Patrick Hayes's "What Is a Computer?" (1997) is a symposium that responds to Searle. Hayes's own view is that a computer is a machine that can take, as input, patterns that describe changes to themselves and other patterns, and that causes the described changes to occur. (A related definition—a computer is a device that "change[s] variable assignments"—is offered in Thomason 2003: 328.) It turns out that it is surprisingly difficult to give a precise characterization of what a computer is.

A closely related topic for which a relevant reading did not appear till after the semester was over is the question of whether the universe itself is a computer (or whether parts of the universe compute; *e.g.*, does the solar system compute Kepler's laws?). On this, see Seth Lloyd and Y. Jack Ng's "Black Hole Computers" (2004). This issue also concerns the nature of simulation (see Rapaport 1998, Perruchet and Vinter 2002 [esp. §1.3.4], and the discussion of programs as scientific theories, below).

What is an algorithm?—Part II. As hard as it is to define 'computer,' the notion of "algorithm" is even murkier, despite the accomplishments of Church, Gödel, Kleene, Markov, Turing, Post, and others. Introductions to computer science often liken algorithms to recipes, and, indeed, there are clear similarities. But the differences are even more illuminating, given the informality with which recipes are usually presented. An interesting unpublished paper by Beth Preston (2000)

suggests that recipes are more like specifications than they are like algorithms. And Carol Cleland has written a series of papers (1993, 1995, 2001, 2002) that explores the relationships between algorithms, recipes, and procedures, introducing a notion of "mundane" procedures (causal processes, including recipes), which are effective procedures that (she argues) are not TM-computable, since their effectiveness depends on the external world.

What is hypercomputation? "Hypercomputation" is a name given by the philosopher Jack Copeland (2002) to the computation of functions that can't be TM-computed. We briefly investigated Turing's "oracle" machines, Putnam's and Gold's "trial and error" machines (Turing machines where it is the *last* answer that counts, not the first answer), Boolos and Jeffrey's infinitely accelerating "Zeus" machines, and Wegner's "interaction" machines (such as automatic-teller machines or airline-reservation systems) (see Copeland 2002 for citations and other models). We also looked at Kugel's (2002) thesis that Putnam-Gold machines may be needed for AI to succeed.

What is a computer program? We focused on five aspects of this question: the nature of implementation, whether programs are theories, the nature of software (vs. hardware), whether software can or should be copyrighted or patented, and whether programs can be verified. Each is discussed briefly, below, with a digression on course evaluation.

What is Implementation? "Implementation" is a ubiquitous notion in computer science, but one that is rarely defined, and thus crying out for philosophical analysis. We say that programs implement algorithms, yet high-level programs can be implemented in machine language. We say that particular data structures (*e.g.*, arrays) can implement abstract data types (ADTs) (*e.g.*, stacks), yet some ADTs (*e.g.*, stacks) can be implemented in other ADTs (*e.g.*, linked lists). Is implementation a relation between an abstraction and something "concrete," or can it (also) be a relation between two abstractions? Is it an isomorphism, or a homomorphism? In rebuttal to Searle's argument that everything is a computer (see What is a Computer?—Part II, above), David Chalmers (1994) develops a notion of implementation as isomorphism. I have urged that implementation is best viewed as the semantic interpretation of an abstract formal system (Rapaport 1999 and forthcoming-a). These issues were all touched upon, and I also used this opportunity to carefully develop the notions of syntax, semantics, and formal systems.

Are Programs Scientific Theories? Some computational cognitive scientists (*e.g.*, Pylyshyn 1984: 76, Johnson-Laird 1988: 52) have claimed that cognitive theories are best expressed, not in the languages of statistics or mathematics, or even in natural language, but in computer programs. These programs, being simultaneously theories and models (or implementations of the theories), can then be executed,

in order to test whether the theory is a good model of cognition. It has also been argued, of course, that such a program is more than merely a model or simulation of the cognitive phenomenon under investigation; some have argued that it actually exhibits the cognitive ability. As background, we also discussed the relationships between theories and models, simulations and "the real thing,"and simulations and emulations; philosophical theories of scientific explanation; and philosophical theories of scientific models. Relevant readings here also include Joseph Weizenbaum's *Computer Power and Human Reason* (1976: chaps. 5 and 6 are on models and theories) and Herbert Simon's *Sciences of the Artificial* (1996: chap. 1, which discusses scientific theories, is also good reading for the question of whether computer science is a science).

What is Software?

Introductory computer science courses often assume that the distinction between software and hardware is clear. Computer scientists and philosophers know otherwise. James Moor's "Three Myths of Computer Science" (1978) points out the inadequacies of the usual "abstract" software vs. "concrete" hardware distinction, arguing that software is a computer program that is changeable by a person. This allows for the "software" to be "hardwired," as long as it can be changed. The "software is abstract" point of view is well argued by Peter Suber (1988), who considers it to be "syntactic form" (and this ties in nicely with the discussion of syntax vs. semantics in the section on implementation). Finally, Colburn (1999) views software as a "concrete abstraction": It has a "medium of description" insofar as it is a text in a formal language (which is an abstraction), and it has a "medium of execution" insofar as it is implemented in circuits and semiconductors (which are concrete).

Interlude: Midsemester Course Evaluation and Course Correction

The previous topic brought us more or less to the midsemester point in the course. Borrowing an idea from my colleague Stuart C. Shapiro, I traditionally give a midsemester course evaluation. I strongly recommend this for any course: It is far more useful than an end-of-course evaluation that is not seen until the course is over and hence is of no use in improving the course that just ended. For this course, I asked two simple, open-ended questions: What aspects of the course would you like to see changed? and What aspects of the course do you especially like? The answers let me know what needed to be fixed and

what was going well. I summarized the answers and posted a response to the course newsgroup.

For this course, the major complaint was the amount of reading. I told the students that I would try to comply with their request for less reading, but that there were just so many exciting things that I wanted them to read that I would compromise: From then on, I only assigned one (sometimes two) required readings for each of the remaining topics, per class session, but I recommended (sometimes strongly) other things to look at—if not now, then at their leisure after the semester was over. Thus, for example, instead of requiring the students to read Moor 1978 *and* Suber 1988 (which is a very long paper) *and* Colburn 1999 (which is philosophically challenging), I only *required* them to read Moor 1978 (which is well-written and also discusses other important topics), *strongly recommended* Suber 1988 (which is wide-ranging and has lots of things to think about), and recommended Colburn 1999. In lecture, however, I discussed all three.

I hasten to add that there were many compliments, too! Students were pleased with the topics and organization, and especially liked the writing assignments, which I discuss below.

Can Software Be Patented? Or Should it Be Copyrighted? The topic of whether computer programs are copyrightable entities or patentable entities[15] is a fascinating one, because it combines legal, social, and metaphysical issues. We concentrated on the last of these, since it flows nicely from the previous topic of what software is.

Here is the fundamental paradox: If a computer program is viewed as a written text, then it is, by definition, copyrightable. But the very "same" program, engraved on a CD-ROM and, hence, executable on a computer, can be viewed as a machine that is, by definition, patentable (as well as subject to legal limitations on exportation to foreign countries; see Colburn 1999). Yet, also by definition, nothing is both copyrightable and patentable. (Whether one *should* copyright or patent a program vs. whether programs should be "open source" is one of the interesting social issues that we did not have time to consider.)

We looked at the legal definitions of copyright and patent (available from various US government websites)[16] and read a fascinating—and little known—essay by computer scientist Allen Newell ("The Models are Broken, the Models are Broken") that appeared as part of a symposium on this topic in the *University of Pittsburgh Law Review* (1985–1986). Newell argues that we computer scientists need to devise better models—*i.e.*, better ontological theories—of such computer-science entities as algorithms, programs, etc. In contrast, some legal scholars (*e.g.*, Koepsell 2000) have argued that lawyers need to devise better methods of legal protection that better match the unique natures of computer software and hardware. The point in both cases is that

there is a mismatch between computer-science entities, on the one hand, and legal forms of protection, on the other (or between computational ontology and legal ontology); something's got to give.

Can Programs Be Verified? We ended our investigations into the nature of computer programs with an inquiry into whether they can be formally verified. There is a subfield of computer science and software engineering that looks into formal methods for proving program correctness (see, *e.g.*, Gries 1981 for a classic treatment). Two philosophers have written essays that critique this approach. I am a firm believer in the value of such formal proofs (despite some very real limitations), and I have several times taught our department's course on program verification. Consequently, I spent some time introducing some aspects of formal verification before turning to the criticisms.

Brian Cantwell Smith's (1985) "Limits of Correctness in Computers" is, in my opinion, one of the most significant papers on all aspects—moral, legal, semantic, ontological, etc.—of the philosophy of computer science, and should be required reading for all computer science majors. Among other things, he argues that there is a gap between the world and our models of it and that computers are doubly removed, relying on models of the models, yet must act in the real world.

The other critique is James Fetzer's explosive essay, "Program Verification: The Very Idea," that appeared in the *Communications of the ACM* in 1988 and that launched a vicious public debate on the pros and cons of verification. Briefly, Fetzer argues that *programs* can't be verified because you can't logically prove that causal systems won't fail; at best, you can verify an *algorithm*. Note that, in order to properly evaluate Fetzer's argument, you must have a firm grasp of the relationship of algorithm to program, which, by this time, my students were well prepared for.

Philosophy of AI: *Could* we build artificial intelligences? As I indicated above, the philosophy of AI deserves a full course to itself (see, *e.g.*, Moulton and Voytek 1979, Rapaport 1986), and one of my motivations for creating a course in the philosophy of computer science (and not merely the philosophy of AI) was that there were many non-AI philosophical issues of interest. Nevertheless, the philosophy of AI is a proper part of the philosophy of computer science, it is my own area of expertise, and the students intensely wanted to discuss it.

I limited myself to two topics: the Turing Test and the Chinese-Room Argument. A case can be made that an excellent course on the philosophy of computer science could consist solely of close readings of Turing's two major essays: his 1936 paper on computability and his 1950 paper on whether computers can think. So, for this topic, we read Turing's "Computing Machinery and Intelligence" (1950) as well as the

current (and probably perennially most popular) reply: John Searle's Chinese-Room Argument (Searle 1980).

Turing 1950, as is well known, argued that a computer will be said to be able to think if we cannot distinguish its linguistic (hence cognitive) behavior from a human's. Searle 1980 proposed a now-classic counterexample that alleges that a computer could pass a Turing Test without really being able to think.[17] (A good source for both of these, and related, papers is Shieber 2004; cf. Rapaport, forthcoming-b.) We closed this topic with my own attempt at a rebuttal of Searle (Rapaport 2000), arguing that syntactic symbol manipulation of the sort that computers do can suffice for semantic interpretation of the kind needed for computational cognition.

Computer ethics. Our final topic was computer ethics. As noted above, and as with philosophy of AI, this is often the topic of full courses by itself and is the subject of numerous texts and anthologies. I gave a brief overview of (computer) ethics, based on Moor's "What Is Computer Ethics?" (1985). We focused on his claim that we need to have metaphysical and ontological theories of computers (in particular, their "logical malleability") and related phenomena in order to answer ethical and social questions about their nature and use.

I chose to concentrate on two issues that are not often covered in such courses or books: Are there decisions that computers should never make? and Should we build artificial intelligences?

We turned to Moor's "Are There Decisions Computers Should Never Make?" (1979). One of his main points is that there are no decisions computers shouldn't make, at least as long as their track record is better than that of humans, but it's up to us to accept or reject their decisions. An interesting contrasting opinion is that of Friedman and Kahn's "People Are Responsible, Computers Are Not" (1992), which argues that there *are* decisions that computers should not make, because only humans are capable of being moral agents. But "to err is human," and we looked at a recent case of an airline crash caused by following a human's decision instead of a computer's (as reported in George Johnson's "To Err Is Human," 2002).

On ethical issues in AI, we read Michael R. LaChat's "Artificial Intelligence and Ethics: An Exercise in the Moral Imagination" (1986). First, I outlined the plot of Stanislaw Lem's "Non Serviam" (1971)—which should be required reading for all researchers in artificial life!—in which what we would today call an Artificial Life researcher is forced to pull the plug on his creations when his research grant ends. LaChat considers whether such research shouldn't even *begin* but agrees that considering the possibilities enables us to deal with important issues such as: What is a person? Would an AI with personhood have rights? Could it be moral?

Philosophy of computer science: A summary and a unifying theme. In closing the semester, I asked the students to read two recent overviews of issues in the philosophy of computer science, as a way to gauge what they had learned—Matthias Scheutz's "Philosophical Issues about Computation" (2002) and Smith's "The Foundations of Computing" (2002)—and we reviewed the semester's readings and discussion, with an eye towards themes that connected the several topics.

One such theme that the students and I became aware of as the semester progressed is the relation of an abstract computation to the real world. This theme is addressed explicitly in some of the papers we read, and is implicit in many others. It emerges in Cleland's discussion of the causal nature of "mundane" procedures, which produce some actual product or physically affect the real world in some way. This is also one of Smith's themes in his "Limits of Computation" essay, as well as an underlying reason of Fetzer's arguments against program verification. It is, of course, the subject matter of implementation, and underlies the paradoxical nature of software vs. hardware, and hence the issue of whether software is copyrightable or patentable. I recommend an exploration of this theme as a unifying idea for a future course in philosophy of computer science.

Assignments

A difficulty. I wanted the students to do a lot of reading and thinking. Thinking is best done by active reading (Rapaport 2005a), discussion, and writing—lots of writing. There is a well-known drawback to assigning a lot of writing to students: The instructor has to read it all and, ideally, comment on it. When this course was first advertised, I expected about ten to fifteen students, in a small seminar-like setting. The first preliminary enrollment report said that thirty students had signed up. Thinking that they thought that this might be a "gut" course ("A philosophy course in a computer science department? Oh, this'll be easy to ace!"), I posted a note to the undergraduate newsgroup spelling out the large quantities of writing that I would expect. Enrollment doubled to sixty! It finally settled down at just under fifty students.[18] Still, fifty ten-page term papers plus frequent short writing assignments during the semester was not a prospect that I looked forward to.

Nor could I rely on help from graduate teaching assistants or recitation sections (a problem I was familiar with from my days teaching at a primarily undergraduate institution). No recitation sections had been assigned to the course, since I had not expected such a large enrollment. They would have been useful for discussion purposes, but that was not to be. I did have an excellent graduate teaching assistant, but he was a computer-science grad student, not a philosophy grad

student (although he did have some undergraduate philosophy experience and was philosophically sophisticated), and, in any case, he had no recitation sections to lead. Consequently, he was of most use to me in keeping records, though he did hold office hours and students felt comfortable going to him for advice on writing.

But how to have students write a lot without having to read it all? And how to have discussions without special time allotted for them? Of course, faculty at undergraduate institutions face this problem all the time, unlike we faculty at research universities. And so I drew upon my experiences as a philosophy professor at an undergraduate institution with no TAs and no recitation sections.

A solution: Required, short position papers . . . I assigned the students five one-page position papers throughout the semester, roughly one every two or three weeks. A first draft of each assignment was due one week after it was announced. The day it was due we set aside for "peer editing" (adapted from techniques used in English composition classes; cf. Cho and Schunn 2004): Students were asked to bring five copies of their position papers, one for me, one for themselves, and one each for three other students. I put the students into small groups of three or four "peers," each of whom had written a response to the same assignment. I asked them to spend about ten to fifteen minutes on each paper, reading it, critiquing it, and making suggestions for improvement. The students were then given another week to revise their papers to be handed in for credit.[19] To ease my burden of grading, I read and put copious comments on only about 40 percent of the papers for each of the five assignments; each student received at least two papers fully critiqued by me (the other three papers were recorded as being handed in).

Peer editing accomplished several goals simultaneously: The students had plenty of opportunities to discuss the material with each other. In fact, probably more students participated in these small groups than would have ordinarily spoken out in a large classroom setting (though such full-class discussions were encouraged, as well). Moreover, all students got multiple feedback on each paper, in addition to my feedback on a subset of their papers. Another advantage of peer editing in class is that I had the freedom (and responsibility) to walk around the room, listening to the student discussions and occasionally facilitating one or answering a question on a one-on-one basis.

The position papers were designed to supplement the lectures and readings, as well as to foster the students' critical-thinking skills. In particular, the topics always involved an argument that the students were asked to evaluate in terms of factuality (*i.e.*, truth value of the premises) and validity (*i.e.*, reasoning). Students were encouraged to present their opinions and to support them with reasons. As one ex-

ample, Position Paper 1, on "What is computer science?" asked the students to evaluate the following argument (embedded in the context of a story about a dean moving a computer science department from a school of science to a school of engineering):

1. Science is the systematic observation, description, experimental investigation, and theoretical explanation of natural phenomena.
2. Computer science is the study of computers and related phenomena.
3. Therefore, computer science is not a science.

(All assignments and peer-editing guidelines are on the Web at http://www.cse.buffalo.edu/~rapaport/510/pospapers.html.)

As one student observed later, the argument-analysis format of the position papers made them somewhat easier to grade than an ordinary essay would have been. Since the students were required to examine a rigid structure of an argument, they had fewer "degrees of freedom" in writing their responses. Thus, grading such papers can be closer to grading a mathematical problem set than a typical essay. It also made grading somewhat more impartial and somewhat less controversial.[20]

. . . **And two optional assignments.** In addition to the required position papers, there was an optional term paper, whose topic had to be approved by me in advance. I supplied a list of some possible topics, but I encouraged the students to explore areas of interest to them. As a default topic, a student could write a summary of the philosophy of computer science in the style of an encyclopedia article or else present his or her own answers to the syllabus questions (see "Syllabus" above). An exclusive-alternative option was a final exam (students could do the exam or the term paper, but not both). This was a take-home, short-answer, essay-style exam, asking for analytic and evaluative summaries of the possible answers to the topic-questions.

A required reading journal. In addition, in order to provide evidence that the students were really reading the material, as well as to encourage them to read slowly and actively, I required them to keep a "reading journal." For each essay they read, they were to copy interesting passages (or at least full references to them) and—most importantly—to provide their own comments on them and on the issues raised in each item read. (Suggestions on how to do this can be found in Rapaport 2005a.) I collected these Journals at the end of the semester, and included them in the grade calculation.

Students who attended almost all classes and turned in a Reading Journal could get a C; students who did that plus all five position papers could get a B; and students who did all of that plus either the term paper or final exam could get an A. All but one student wrote the

position papers. Over 80 percent of the students chose the exam/paper option, with about 70 percent choosing the exam option.

What the Students Did and Didn't Like

The students' favorite part of the course was the writing, peer-editing, and revising of the one-page position papers: They enjoyed the discussions, the ability to revise (including an option to re-revise for a higher grade), and—most importantly—the skills they learned, and the practice they got, in critically analyzing and evaluating informal arguments. In addition, well after the course ended, some students told me that they have continued keeping reading journals in their other courses and research.

They also appreciated the course website, which has links to the syllabus and a directory of documents that, in turn, has a large bibliography, links to other relevant websites, and links to the assignments, position papers, term-paper topics, and final exam. I began each new section of the course by putting up a webpage containing recommended readings for that topic. I then gave a quick overview in lecture about each of the readings. Students informed me that this was very useful because it provided a summary of what was to come, including the different positions that have been taken on each issue.

Here is what one student said, in an unsolicited e-mail message I received after the course was over:

> I'd like to thank you for putting together such a great course this semester. I'll admit, knowing very little about it, I never had much respect for philosophy in the past—but this course has provided me with an entirely new perspective. In fact, I'd say that I learned as much in your course as any other I've taken in my graduate career at UB (not to mention the fact that the skills I learned in [it] are far more transferable than the skills of the more esoteric CS courses). . . . I urge [you] to offer this course again in the future. It offers exactly the kind of breadth of education that the department needs to stress, and with its CS flavor, it can tap the interest of students who would otherwise blow it off. Thanks again for a great semester, and please consider making Philosophy of CS a regular offering :)

Another student observed that "I don't think there was a single student in the class whose critical thinking/writing/reading skills didn't improve as a result of taking this course."

As noted above, the students' least favorite part of the course was the amount of reading. Of course, this is something that students almost always complain about, but, in this case, the complaint really was about the quantity, not the quality: By and large, they found all of the readings to be interesting and useful; their complaint was that they didn't have enough time to read them all as carefully as they (and I) would have liked. Fortunately, on the basis of the mid-semester course evalu-

ation, I found this out early enough to be able to do something about it. As discussed above, subsequent reading assignments were limited to at most two required readings, with suggestions for recommended (but optional) follow-up readings.

Conclusions

I believe this to have been a worthwhile course, both for me and—more importantly—for the students. It gave many of the computer science majors the option to think about many issues that they either hadn't thought of before or had thought about but had no venue for discussing. It also gave them an opportunity to (learn how to) think critically, and to find out what philosophy could be like. The philosophy majors, in addition, had the opportunity to learn some things about computers, computing, and computer science that they probably would not have come across in more traditional philosophy courses, as well as the opportunity to apply some of their philosophical skills and knowledge to a different domain.

Notes

I am grateful to my students Dima Dligach and Albert Goldfain, to my colleagues Peter D. Scott and Stuart C. Shapiro, and to an anonymous reviewer for comments on earlier drafts.

1. In particular, CD5650, Swedish National Course on Philosophy of Computer Science, at Mälardalen University (Sweden), coordinated by Gordana Dodig-Crnkovic (http://www.idt.mdh.se/~gdc/PI-network-course.htm); Selected Topics in the Philosophy of Computer Science, at Tel Aviv University (Israel), taught by Eli Dresner (http://www.tau.ac.il/humanities/digicult/english.htm); and PHI 319, Philosophy of Computing, at Arizona State University, taught by Bernard W. Kobes (http://www.asu.edu/clas/philosophy/course_descripts.htm).

2. I am grateful to Timothy Colburn, Randall R. Dipert, Eli Dresner, James H. Fetzer, Luciano Floridi, Bipin Indurkhya, James Moor, Robert Stainton, and Chris Viger for (e-mail) discussions on the questions that such a course might focus on.

3. The home page for the course, with links to the complete syllabus, assignments, and other course web pages, is at http://www.cse.buffalo.edu/~rapaport/philcs.html and archived as Rapaport 2005b.

4. Pointers to these and other sources are at my course Web page "What is Philosophy of Computer Science?" (http: //www.cse.buffalo.edu/~rapaport/510/whatisphilcs.html).

5. This claim is based on a not unreasonable assumption that computer-science students tend to be "Dualists" who see (and fear?) philosophy as being a "Multiplistic" discipline. These are terms from William Perry's (1970, 1981) "scheme" of intellectual and ethical development. For a quick online glimpse of Perry's views, see my website, "William Perry's Scheme of Intellectual and Ethical Development" (http://www.cse.buffalo.edu/~rapaport/perry.positions.html).

Roughly, "Dualists" believe that all questions have correct answers and that the student's job is to learn these answers, whereas "Multiplists" believe that most questions have no known answers and, consequently, that everyone's opinion is equally

good. However, those are vast oversimplifications, and the interested reader is urged to consult Perry's writings, or any of the other sources listed on my website. On whether there can be answers to philosophical questions and, thus, real progress in philosophy, see Rapaport 1982.

6. In Perry's terminology, philosophy is a "Contextually Relativistic" discipline, *i.e.*, one that critically evaluates claims on the basis of evidence (the truth-value of a claim is "relative" to its evidential "context").

7. Learned from my former philosophy colleague, Kenneth G. Lucey.

8. There are many excellent textbooks on critical thinking and informal logic, and, of course, it is the subject of many full courses on its own. A useful short introduction for a course such as this is Longview Community College's website "Critical Thinking Across the Curriculum Project" (http://www.kcmetro.cc.mo.us/longview/ctac/toc.htm).

9. Another (more recent) view is that computer science is the study of *virtual* phenomena (Crowcroft 2005).

10. Some of whom had their doctorates from departments of computer *science*!

11. http: //www.cse.buffalo.edu/~rapaport/510/history.html.

12. http: //www.computer.org/computer/timeline/.

13. This is an adaptation of Stuart C. Shapiro's informal characterization; personal communication.

14. On this sort of Talmudic, slow-but-active reading style, see Rapaport 2005a, §5.

15. There is a third possibility: that they are trademarkable entities; we did not consider this option.

16. For 'copyright,' see the US Copyright Office Circular 1 at http://www.copyright.gov/circs/circ1.html#wci; for 'patent,' see the US Patent and Trademark Office Glossary at http://www.uspto.gov/main/glossary/index.html#p.

17. Mention should be made that a very early version of Searle's thought experiment appears as a way of explicating Turing machines in Rogers 1959 (part 1, reprinted in 1969: 131, 133; based on a 1957 lecture).

18. The breakdown of student majors was as follows:

	CSE	PHI	Other
undergrads	72%	10%	12%
grads	75%	15%	10%
total	73%	12%	15%

CSE = Computer Science and Engineering majors; PHI = Philosophy majors; Other = students majoring in Biology, Economics, Electrical Engineering, Management, Management and Information Science, and Mathematics.

19. My writing guidelines and a brief guide to grammar and punctuation are on the Web at http://www.cse.buffalo.edu/~rapaport/howtowrite.html.

20. For more thoughts on grading, and a "triage" theory of grading, see my website, "How I Grade," http://www.cse.buffalo.edu/~rapaport/howigrade.html.

Bibliography

Alexander, Peter. 1967. "Mach, Ernst," in *Encyclopedia of Philosophy*, ed. Paul Edwards, vol. 5: 115–19.

Audi, Robert. 2001. "Philosophy: A Brief Guide for Undergraduates" http://www.apa.udel.edu/apa/publications/texts/briefgd.html.

Brooks, Frederick P., Jr. 1996. "The Computer Scientist as Toolsmith II," *Communications of the ACM* 39:3 (March): 61–68.

Chalmers, David J. 1994. "On Implementing a Computation," *Minds and Machines* 4 (1994): 391–402.

Cho, Kwangsu, and Christian D. Schunn. 2004. "You Write Better When You Get Feedback from Multiple Peers than from a Single Expert," *Proceedings of the 20th Annual Conference of the Cognitive Science Society* (Mahwah, N.J.: Lawrence Erlbaum Associates, 2005).

Cleland, Carol E. 1993. "Is the Church-Turing Thesis True?" *Minds and Machines* 3:3 (August): 283–312.

_____. 1995. "Effective Procedures and Computable Functions," *Minds and Machines* 5:1: 9–23.

_____. 2001. "Recipes, Algorithms, and Programs," *Minds and Machines* 11:2 (May): 219–37.

_____. 2002. "On Effective Procedures," *Minds and Machines* 12:2 (May): 159–79.

Colburn, Timothy R. 1999. "Software, Abstraction, and Ontology," *The Monist* 82:1: 3–19; reprinted (in slightly different form) in Colburn 2000, chap. 12.

_____. 2000. *Philosophy and Computer Science* (Armonk, N.Y.: M. E. Sharpe).

Copeland, B. Jack. 2002. "Hypercomputation," *Minds and Machines* 12:4: 461–502.

Crowcroft, Jon. 2005. "On the Nature of Computing," *Communications of the ACH* 48:2: 19–20.

Davis, Martin, ed. 1965. *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems, and Computable Functions* (New York: Raven Press).

Davis, Michael. 1998. *Thinking Like an Engineer: Studies in the Ethics of a Profession* (New York: Oxford University Press).

Denning, Peter J. 1985. "What Is Computer Science?" *American Scientist* 73 (January–February): 16–19.

Ensmenger, Nathan. 2004. "Bits of History: Review of A. R. Burks's *Who Invented the Computer? The Legal Battle that Changed Computing History*," *American Scientist* 91 (September–October): 467–68.

Fetzer, James H. 1988. "Program Verification: The Very Idea," *Communications of the ACM* 31:9 (September): 1048–63; reprinted in *Program Verification: Fundamental Issues in Computer Science*, ed. Timothy R. Colburn, James H. Fetzer, and Terry L. Rankin (Dordrecht: Kluwer Academic Publishers, 1993), 321–58.

Floridi, Luciano. 1999. *Philosophy and Computing: An Introduction* (London: Routledge).

_____. 2004. *The Blackwell Guide to the Philosophy of Computing and Information* (Malden, Mass.: Blackwell).

Friedman, Batya, and Peter H. Kahn, Jr. 1992. "People Are Responsible, Computers Are Not," excerpt from their "Human Agency and Responsible Computing: Implications for Computer System Design," *Journal of Systems Software* (1992): 7–14; excerpt reprinted in *Computers, Ethics, and Society*, second edition, edited by M. David Ermann, Mary B. Williams, and Michele S. Shauf (New York: Oxford University Press, 1997), 303–14.

Gries, David. 1981. *The Science of Programming* (New York: Springer-Verlag).

Hartmanis, Juris, and Herbert Lin. 1992. "What Is Computer Science and Engineering?" in *Computing the Future: A Broader Agenda for Computer Science and Engineering*,

ed. Juris Hartmanis and Herbert Lin (Washington: National Academy Press), chap. 6, pp. 163–216.

Hayes, Patrick J. 1997. "What Is a Computer? An Electronic Discussion," *The Monist* 80:3.

Henkin, Leon. 1962. "Are Logic and Mathematics Identical?" *Science* 138:3542 (November 16): 788–94.

Herman, Gabor T. 1983. "Algorithms, Theory of," in *Encyclopedia of Computer Science and Engineering*, second edition, edited by Anthony S. Ralston (New York: Van Nostrand Reinhold), 57–59.

Johnson, George. 2002. "To Err Is Human," *New York Times* (14 July).

Johnson-Laird, Philip N. 1988. *The Computer and the Mind: An Introduction to Cognitive Science* (Cambridge, Mass.: Harvard University Press).

Kemeny, John G. 1959. *A Philosopher Looks at Science* (Princeton, N.J.: D. van Nostrand).

Knuth, Donald. 1974. "Computer Science and Its Relation to Mathematics," *American Mathematical Monthly* 81:4 (April): 323–43.

Koepsell, David R. 2000. *The Ontology of Cyberspace: Philosophy, Law, and the Future of Intellectual Property* (Chicago: Open Court).

Kugel, Peter. 2002. "Computing Machines Can't Be Intelligent (. . . and Turing Said So)," *Minds and Machines* 12:4: 563–79.

LaChat, Michael R. 1986. "Artificial Intelligence and Ethics: An Exercise in the Moral Imagination," *AI Magazine* 7:2: 70–79.

Lem, Stanislaw. 1971. "Non Serviam," in S. Lem, *A Perfect Vacuum*, trans. Michael Kandel (New York: Harcourt Brace Jovanovich, 1979).

Lloyd, Seth, and Y. Jack Ng. 2004. "Black Hole Computers," *Scientific American* 291:5 (November): 52–61.

Loui, Michael C. 1987. "Computer Science Is an Engineering Discipline," *Engineering Education* 78:3: 175–78.

_____. 1995. "Computer Science Is a New Engineering Discipline," *ACM Computing Surveys* 27:1 (March): 31–32.

Moor, James H. 1978. "Three Myths of Computer Science," *British Journal for the Philosophy of Science* 29:3 (September): 213–22.

_____. 1979. "Are There Decisions Computers Should Never Make?" *Nature and System* 1: 217–29.

_____. 1985. "What Is Computer Ethics?" *Metaphilosophy* 16:4 (October): 266–75.

Moulton, Janice, and Jane Voytek. 1979. "Can Humans Think Machines Think?" *Teaching Philosophy* 3:2: 153–67.

Newell, Allen. 1985–1986. "Response: The Models Are Broken, the Models Are Broken," *University of Pittsburgh Law Review* 47: 1023–31.

Newell, Allen, Alan J. Perlis, and Herbert A. Simon. 1967. "Computer Science," *Science* 157:3795 (22 September): 1373–74.

Newell, Allen, and Herbert A. Simon. 1976. "Computer Science as Empirical Inquiry: Symbols and Search," *Communications of the ACM* 19:3 (March): 113–26.

O'Connor, J. J., and E. F. Robertson. 1998. "Charles Babbage," http: //www-gap.dcs .st-and.ac.uk/~history/Mathematicians/Babbage.html.

Papineau, David. 1996. "Philosophy of Science," in *The Blackwell Companion to Philosophy*, ed. Nicholas Bunnin and E. P. Tsui-James (Oxford: Blackwell), 290–324.

Perruchet, Pierre, and Annie Vinter. 2002. "The Self-Organizing Consciousness," *Behavioral and Brain Sciences* 25:3 (June): 297–388.

Perry, William G., Jr. 1970. *Forms of Intellectual and Ethical Development in the College Years: A Scheme* (New York: Holt, Rinehart, and Winston).

_____. 1981. "Cognitive and Ethical Growth: The Making of Meaning," in Arthur W. Chickering and Associates, *The Modern American College* (San Francisco: Jossey-Bass), 76–116.

Petroski, Henry. 2003. "Early Education," *American Scientist* 91 (May–June): 206–09.

Preston, Beth. 2000. "Recipes and Songs: Towards a Theory of Production" (unpublished ms.)

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

Rapaport, William J. 1982. "Unsolvable Problems and Philosophical Progress," *American Philosophical Quarterly* 19: 289–98.

_____. 1986. "Philosophy of Artificial Intelligence: A Course Outline," *Teaching Philosophy* 9: 103–20.

_____. 1998. "How Minds Can Be Computational Systems," *Journal of Experimental and Theoretical Artificial Intelligence* 10: 403–19.

_____. 1999. "Implementation Is Semantic Interpretation," *The Monist* 82:1: 109–30.

_____. 2000. "How to Pass a Turing Test: Syntactic Semantics, Natural-Language Understanding, and First-Person Cognition," *Journal of Logic, Language, and Information*, 9:4: 467–90; reprinted in *The Turing Test: The Elusive Standard of Artificial Intelligence*, ed. James H. Moor (Dordrecht: Kluwer, 2003), 161–84.

_____. 2005a. "How to Study," http://www.cse.buffalo.edu/~rapaport/howtostudy.html.

_____. 2005b. "Philosophy of Computer Science: An Introductory Course," *Technical Report 2005–16* (Buffalo: SUNY Buffalo Department of Computer Science and Engineering); pre-print version of this article, containing archived webpages; available at http: //www.cse.buffalo.edu/tech-reports/2005-16.pdf.

_____. Forthcoming-a. "Implementation Is Semantic Interpretation: Further Thoughts," *Journal of Theoretical and Experimental Artificial Intelligence.*

_____. Forthcoming-b. Review of Shieber 2004, *Computational Linguistics.*

Rogers, Hartley, Jr. 1959. "The Present Theory of Turing Machine Computability," *Journal of the Society for Industrial and Applied Mathematics* 7: 114–30; reprinted in *The Philosophy of Mathematics*, ed. Jaakko Hintikka (London: Oxford University Press, 1969), 130–46.

Scheutz, Matthias. 2002. "Philosophical Issues about Computation," *Encyclopedia of Cognitive Science* (London: Macmillan).

Searle, John R. 1980. "Minds, Brains, and Programs," *Behavioral and Brain Sciences* 3: 417–57.

_____. 1990. "Is the Brain a Digital Computer?" *Proceedings and Addresses of the American Philosophical Association* 64: 21–37.

Shapiro, Stuart C. 2001. "Computer Science: The Study of Procedures," http://www.cse.buffalo.edu/~shapiro/Papers/whatiscs.pdf.

Shieber, Stuart M. 2004. *The Turing Test: Verbal Behavior as the Hallmark of Intelligence* (Cambridge, Mass.: MIT Press).

Simon, Herbert A. 1996. *The Sciences of the Artificial, Third Edition* (Cambridge, Mass.: MIT Press).

Simon, Herbert A., and Allen Newell. 1958. "Heuristic Problem Solving: The Next Advance in Operations Research," *Operations Research* 6:1 (January–February): 1–10.

Smith, Brian Cantwell. 1985. "Limits of Correctness in Computers," *Technical Report CSLI-85-36* (Stanford, Calif.: Center for the Study of Language and Information); first published in *Computerization and Controversy*, ed. Charles Dunlop and Rob Kling (San Diego: Academic Press, 1991), 632–46; reprinted in *Program Verification: Fundamental Issues in Computer Science*, ed. Timothy R. Colburn, James H. Fetzer, and Terry L. Rankin (Dordrecht: Kluwer Academic Publishers, 1993), 275–93.

_____ . 2002. "The Foundations of Computing," in *Computationalism: New Directions*, ed. Matthias Scheutz (Cambridge, Mass.: MIT Press), 23–58.

Soare, Robert I. 1996. "Computability and Recursion," *Bulletin of Symbolic Logic* 2:3 (September): 284–321.

Suber, Peter. 1988. "What Is Software?" *Journal of Speculative Philosophy* 2:2: 89–119.

Thomason, Richmond H. 2003. "Dynamic Contextual Intensional Logic: Logical Foundations and an Application," in *CONTEXT 2003, Lecture Notes in Artificial Intelligence 2680*, ed. P. Blackburn et al. (Berlin: Springer-Verlag), 328–41.

Turing, Alan M. 1936. "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society*, ser. 2, vol. 42: 230–65.

_____ . 1950. "Computing Machinery and Intelligence," *Mind* 59: 433–60; reprinted in Shieber 2004.

Weizenbaum, Joseph. 1976. *Computer Power and Human Reason: From Judgment to Calculation* (New York: W.H. Freeman).

Woodhouse, Mark B. 2003. *A Preface to Philosophy*, 7th edition (Belmont, Calif.: Wadsworth/Thomson Learning).

William J. Rapaport, Department of Computer Science and Engineering, Department of Philosophy, and Center for Cognitive Science, State University of New York at Buffalo, Buffalo NY 14260-2000, rapaport@cse.buffalo.edu, http://www.cse.buffalo.edu/~rapaport/