The Rigor Resolution on Undergraduate Education

proposed for consideration by the

Department of Computer Sciences
University of Texas at Austin

by
Robert S. Boyer, Professor

September, 1995

Preamble

Computer science is a mathematical rather than a physical science. Following Church's thesis, we believe that the class of computations based on any currently imagined digital technology is completely characterized by the mathematical objects known as the partial recursive functions. That is, computer scientists need not make observations and experiments to determine the laws of the physical world relevant to their discipline; rather, computer scientists already know the fundamental law of computing, namely that we can compute exactly what can be computed by a universal Turing machine.

Ideally, an undergraduate computer science curriculum should take as its principal goal that the students become skilled in reasoning rigorously about computing. And just as mathematics majors are taught rigorous mathematical thinking entirely by the method of rigorously proving theorems about specific objects such as groups and continuous functions, computer science undergraduates should be taught rigorous mathematical thinking exclusively by the method of rigorously proving theorems about specific computational objects, such as specific partial recursive functions, i.e., algorithms.

Unfortunately, our computer science undergraduate curriculum has been so driven by concerns unrelated to science that the scientific component has been unreasonably neglected. Principal among these unscientific concerns is the desire to provide students with skills which we or they think will most suit them for jobs in the computing industry. This concern with employment has resulted in a very bad effect upon the undergraduate computer science

curriculum: the students receive a vast but sloppy and unrigorous knowledge of many programming language constructs, architectures, and systems. How many of our undergraduate students, even at the conclusion of their education, can assert with perfect accuracy and defend with perfect logic any interesting scientific statements about these many constructs, architectures, and systems to which they have been exposed? I conjecture that the answer is: "Essentially none." Even if such an education is producing students who can easily find jobs, this policy of rapid, sloppy exposure to many aspects of computing, all poorly learned, is spreading a thesis that pollutes not only our students' minds but also the entire future of computing—that there are no real truths in computing, no science, only layers of partial, vague rules of thumb among which trial and error is preeminent.

As an antidote to what I perceive as a great overemphasis upon unrigorous teaching in computing, I propose, by the following four resolutions, a major change of direction in our undergraduate curriculum.

The Rigor Resolution

RR-1. Resolved, that the Computer Sciences Department takes it as an objective, over the next ten years, to revise completely the undergraduate curriculum so that the following result is obtained, to wit, that every course in computing shall be taught upon a strict, mathematical basis. In every case that a computing system, language, architecture, algorithm, or technique is discussed, it will be presented to (or developed by) the students in a strictly rigorous fashion. Any program or system developed in such a class shall be developed in such a way that "correct" has a strictly mathematical, proven meaning. For example, the program or system may be proved to satisfy precisely given functional or performance requirements. This rigor requirement shall be extended to any prerequisite course that we require a student to take outside of computer science.

RR-2. Resolved, that those faculty members most skilled and experienced in reasoning rigorously about programs, and who also have some experience with undergraduate teaching, shall be mainly responsible for designing and teaching the introductory courses. A skill is best learned from the best. A skill badly learned is almost impossible to unlearn.

RR-3. Resolved, that in systematically reconsidering each and every course in the undergraduate curriculum, we shall insist upon identifying and publishing an answer to the question "What precise, incontrovertible scientific propositions are stated and proved in this course?"

RR-4. Resolved, that the primary societal objective we shall pursue in educating our undergraduate majors will be to prepare students for admission to and success in first rate graduate computer science programs, with the hope that these students will go on to advance the science of computing. (We take it that an analogous objective is currently followed in the departments of the two paradigm sciences, mathematics and physics.)

Some Discussion Points

1. *Whatever do you mean by* rigor*?* Rigor is admittedly a vague term with differing meanings. We have, from the Frege-Whitehead-Russell-Gödel tradition, the idea of a formal mathematical proof. Even though very few people today produce formal proofs, by *rigorous* I mean an argument for which the advocate has a "well-informed belief" that it is possible to transform the argument into a formal one, given time and incentive to do so. The degree of rigor used in mathematics undergraduate education increases between freshman and senior level courses, and I think it would be sufficient if our freshman courses were as rigorous as a good freshman calculus course, the kind that spends some time on definitions and proofs, not merely on unjustified symbolic manipulations. Our junior and senior courses should be as rigorous as junior and senior math courses in algebra and analysis.

2. *What do you do with faculty who have no interest or ability in teaching such rigorous courses?* I believe that we have quite a good department, and I doubt that anyone in our department is incapable of giving rigorous courses. But as a strong advocate of academic freedom, I have no intention of attempting to *force* anyone to teach certain subjects or to *force* them to teach in a certain way. On the contrary, I am hoping by this resolution to help develop our understanding of what computer science *is*. I suspect that if by some miracle a majority of the members of our department were to agree with this resolution, then a high percentage of the dissenting faculty members would conscientiously, gladly, and voluntarily adapt their teaching to suit.

3. *Insisting upon the universal use of formal methods would have the unfortunate effect of abolishing intuition from the classroom.* I want to make it clear that although I am, by this resolution, insisting upon rigor, I am not insisting upon formality. While we may hope for the day in which students learn to derive programs as correctly as they learn to derive results by long division, I am not yet personally convinced that such formal methods as we have are what we ought to be teaching exclusively. It seems to me that for a while, at least, we must continue to tolerate the idea of conducting mathematical arguments with the informal rigor that has been used at least since Euclid.

4. *You have completely ignored the importance of practical system building experience in the education of a computer scientist. The programming*

laboratory in computer science is as crucial as the laboratory in chemistry. I believe that there is a major difference of opinion here, and I cannot hope in a few words to dislodge the long held opinions of so many on this topic. But let me instead ask: What *scientific principles* are established in our computing laboratories? I have a very good idea of what sorts of *principles* are established in a chemistry or physics lab, e.g., inverse square law or periodic table. But even though I have been a professor of computer science here for a decade, *I have no idea what scientific principles are being taught in our labs*! Frankly, I suspect that what is being taught in our labs is a form of the cardinal virtue of fortitude: we teach our students to remain astutely alert and persistent while enduring the overwhelming and excruciating pain of learning vast, intricate, ever-changing, but largely unintelligible pseudo-formalisms called "systems." Probably this form of fortitude is *essential* for success in coding programs in modern day, "industrial strength" languages to be run under contemporary operating systems! And perhaps this form of fortitude is highly valued by industry. But I do not believe that instruction in this form of fortitude should be an essential part of a *science* curriculum.

5. *In computing, we must be largely concerned with what can be physically realized today, not with what exists only theoretically or mathematically. An operating system or AI system whose only existence is on paper is of little value. Many ideas in computing have been found practically worthless when people tried to implement the ideas. The way to validate an idea in computing is to show, via an implementation, that the idea really* works. I have great personal and technical empathy with some who hold this view of computing. But I believe that such a view properly belongs in a college of engineering rather than in a college of science. (Indeed, I would admit that a substantial part of my own technical work is properly called engineering.) I am not opposed in any way to doing or teaching good engineering. But in science, one seeks to find and state the truths of things quite independently of the possibility of realizing those truths in the competitive products of today. Our department is called the Department of Computer Sciences, not Computer Engineering. Our department is in the College of Natural Sciences, not Engineering.

6. *Is it not important for a student to be exposed to the applications of computing in addition to the principles of computing? For example, in chemistry, should not students be made aware of such important applications as DNA?* Computing seems destined to have almost as many applications as

does the rest of mathematics or as does physics. Maybe computing already does. An undergraduate education contains perhaps 600 hours of classroom instruction in the major, a modest amount of time. Do mathematics majors have the time in an undergraduate program to study applications of mathematics if they are, in the course of a major, to become fully grounded in algebra and analysis? No. Do physics majors have the time in an undergraduate program to study applications of physics if they are, in the course of a major, to become fully grounded in classical mechanics, electro-magneto dynamics, quantum mechanics, and relativity? No. I suspect that for an undergraduate to become fully grounded in rigorous reasoning about computations would inevitably consume every available moment of time in a major, and I suspect that attention to applications could be feasibly fit into a good program only as illustrations. For example, one might attempt to specify and prove the correctness of a simple operating system kernel or a compiler.

7. *The function of a laboratory in computing is not to explore physical laws, as might be done in a physics or chemistry laboratory, but to allow the students to play with computations hands-on, to see real computations, to have experimental fun and thus grow in their understanding of what can be computed.* There is no doubt that playing with computers can be fun, even as addicting a habit as gambling. But I do not know a mathematical principle that is more readily learned by bright undergraduates through playing with physical objects than by simple reading, speaking, and thinking. How much mathematics is learned by looking at physical conic sections? It may be that computers will revolutionize educational practice someday, but it has not happened yet. Let us help our students learn to think rigorously by the old fashioned, known-to-work methods that require no more physical support than chaulkboards and other ancient writing implements. Let us not divert our limited energies by attempting to show, for the first time, that computers can also be useful tools in helping students learn to think rigorously. Frankly, I suspect that the mental and physical chaos associated with using any popular current computing system is rather inimical to the delicate process of learning rigorous thinking—a process that seems to require a great deal of physical and mental peace. When, as at present, our students' first exposure to computing science is via the "user friendly" Macintosh, how can they fail to speculate that in computing, the key scientific principles are "who needs documentation, much less specification?" "do what is meant, not what is precisely requested," "what you see is all you get," "that feature

is coming in the next version," "that's not a bug, it's a feature," and, most damning of all, that it is fair to *lie* when documenting what a feature does in order not to confuse the novice.

8. *Surely you do not mean to require that every assertion mentioned be proved? What about conjectures?* Clearly, there must be room for the discussion of conjectures. For example, the conjecture $P \neq NP$ is one of the most influential propositions in the history of computing. On the other hand, the enumeration of theorems without proof is a potentially dangerous enterprise that should not be encouraged, even if it is sometimes tolerated for theorems whose proofs are extremely deep.

9. *Our current methods of formal logic seem insufficient for expressing all human knowledge about the world, as seems evidenced by the grave difficulties encountered in 35 years of attempts to formalize "common sense" knowledge in artificial intelligence research, not to mention 2300 years of philosophical attempts to improve upon Aristotelean logic in nonmathematical domains. Thus a too rigid emphasis upon current mathematical logic will preclude progress in that part of computer science which aims at building computing systems that deal intelligently with the world.* We have now enjoyed approximately 60 years of stability in the formal foundations of mathematics, i.e., first order logic and set theory. One can hope and pray for advances in logic that will permit computations to employ "common sense" knowledge when dealing with the world. But it seems that every formal logic so far proposed for reasoning about the world (e.g., temporal or circumscriptional) can be easily understood (i.e., modeled) within first order logic with set theory. While enhanced or alternative logics may someday dislodge the current foundations for mathematics, it seems prudent to wait for these logics to become established in the research and then graduate education communities before worrying about their places in the undergraduate program.

10. *Even if it be granted that the scientific principles of computing are mathematical and hence not suitable for empirical verification in laboratory experiment, nevertheless it may be argued that exploring concepts of computing with "hands on" experience permits some, if not all, individuals to gain a deeper insight into the mathematical principles of computing. Thus some practical work is desirable.* It seems to me that there is, indeed, some truth to this position, but I believe that American computer science education has erred so profoundly in the direction of "experimentation" that the establishment of a curriculum somewhere with *no* "fooling around" would be a good

antidote to this national disaster. The dangers of "experimentalism" are profound and they have been documented widely. Foremost among the dangers is the ignorant and arrogant satisfaction to be derived by saying "I've got something here that works, and that *means* a lot more than all your paper studies." This attitude has considerably delayed progress in some fields, for example, artificial intelligence; funding agencies have been bamboozled into supporting projects that merely produce superficial demos rather than scientific results. This attitude encourages certain psychological pathologies, including the gambler-hacker syndrome. This attitude gives moral support to those software vendors whose only documentation is essentially the instruction to "try all the menus until you find something that works." This attitude also has the effect of spreading the theory that computing systems are inherently too difficult to document precisely, much less verify against precise documentation. I would support the use of any computing system for which precise, formal, and complete documentation was available, if ever such a thing becomes available.

11. *The Computer Sciences Department serves and must continue to serve two constituencies, industry and science.* It has been said "No man can serve two masters: for either he will hate the one, and love the other; or else he will hold to the one, and despise the other. Ye cannot serve God and mammon." Is serving both science and the immediate needs of industry possible for our department's undergraduate program? Can the same program both (a) train programmers to be ready to maintain and enhance contemporary industrial computing systems and (b) train students in the rigorous scientific foundations necessary for graduate studies and research that will advance the science of computing? In my view, certainly not. We have here an important question: How do you view the current state of industrial computing? Is it in awful shape, hobbled by a staggeringly large accretion of nonsensical, unintelligible, pseudo-logical constructs, and thus in need of a radically new, thoroughly scientifically based approach? Or is it in 'pretty good shape' something for which gradual, gentle, inevitable, little contributions from science are all that are required? If you think that industrial computing is in as bad a shape as I think it is, then you will probably agree with me that we cannot simultaneously educate students who will become scientists as we educate students to serve the immediate needs of industry. An undergraduate trained to reason carefully about programs is likely to become confused, perhaps even nauseated, when confronted with

typical industrial computing literature. But if we cannot effectively serve both constituencies, which one should we serve? In my opinion, the choice must clearly be science. The University of Texas at Austin is and ought to be the premier public scientific institution for about 1,000 miles in any direction, a very large portion of the inhabitable surface of the earth. It is our *duty* in undergraduate education to produce students who are ready for graduate training as scientists. It is *our* duty because: if we do not do it, no one else *will*, because no one else *can*. There are over 2,000 colleges and universities in this country; we strive to be in the top 20 of these institutions. Surely, the science departments of the top 1% of all the colleges and universities can focus on the needs of science, letting the other 99% worry about the needs of industry.

12. *Computer sciences is an experimental science. One can prove this by noting that the NSF explicitly supports experimental research in computing.* It seems to me that a silly person could argue that even mathematics is an experimental science, abusing the word *experimental*. After all, mathematicians are always trying out new definitions, new theorems, and trying out new proof strategies. Let's call each of the latter abandoned intellectual efforts an experiment! Does this this really mean that mathematics is experimental? I believe not. The two fundamental questions about any science are What truths does it offer and How do we know those truths? In the natural sciences, such as physics, experimentation is a fundamental part of the "knowing" process; the motions of specific physical objects help to confirm or refute proposed theses of physics. I challenge anyone please to tell me an important thesis in the science of computing that was confirmed or refuted by experiment. I simply cannot think of one.

13. *Computer science is really simply a branch of engineering. So you should not apply to it the standards of a science.* I believe that computing is destined to become one of the greatest of the sciences. Perhaps it already has. But currently things are moving very fast and it is hard to tell how things will work out. We see computer science departments in colleges of engineering and in colleges of business, in addition to within colleges of science. I will not be surprised to see something like computer science subdepartments emerge within departments of physics and biology. Who knows what all these entities associated with computing will eventually be named. Computing is destined to have at least as many applications as mathematics. But mathematicians do not attempt the ridiculous task of trying to include all uses of

mathematics, even all serious uses of advanced mathematics, within a mathematics department. Just as we need mathematics departments, we need to have science departments for computing, separate from all of the applications and practical engineering departments that use computing, even in extremely sophisticated ways. It is for such a science department of computing that this essay has been written.