2-1990

The Central Role of Mathematical Logic in Computer Science

J. Paul Myers Jr.
Trinity University, pmyers@trinity.edu

The Central Role of Mathematical Logic in Computer Science

J. Paul Myers, Jr.

Department of Computer Science
Trinity University
San Antonio, Texas 78212

I. Introduction and a Short Tirade

A fascinating and largely unheralded development accompanying the rise of computer science and computer technology has been the increasing applicability of the two most fervently "pure" branches of mathematics: number theory and mathematical logic. Unexpected mere decades ago are present opportunities for number theorists and logicians in this most applied of the mathematical sciences. Indeed, advertisements for both academic and industrial positions in the ACM *Communications*, for example, call for expertise in these fields. Martin Davis expresses the surprise at this new status of mathematical logic:

> When I was a student, even the topologists regarded mathematical logicians as living in outer space. Today the connections between logic and computers are a matter of engineering practice at every level of computer organization. ... Issues and notions that first arose in technical investigations by logicians are deeply involved, today, in many aspects of computer science. [3]

While number theory is largely confined at present to issues of security and cryptography (as well as continuing "pure" activities: e.g., finding the next Mersenne prime), logic has permeated most of the field in both superficial and profound ways. Number theory, like calculus, has only occasional uses in computation; and thus computer scientists appropriately show only occasional interest in it.

But it's astounding that computer scientists are only occasionally interested in learning the many facets, difficulties, and subtleties of logic since it pervades the entire field. In his influential text, *The Science of Programming*, David Gries poignantly confesses:

> The research was fraught with lack of understanding and frustration. One reason for this was that computer scientists in the field, as a whole, did not know enough formal logic. Some papers were written simply because the authors didn't understand earlier work; others contained errors that wouldn't have happened had the authors been educated in logic. ... We spent a good deal of time thrashing, just treading water, instead of swimming, because of our ignorance. With hindsight, I can say that the best thing for me to have done ten years ago would have been to take a course in logic. I persuaded many students to do so, but I never did so myself. [6]

But Gries' plea has been largely unheeded; logic is not a true part of the computer science curriculum. The requirement of Trinity University's Department of Computer Science -- including formal logic (taught in the Philosophy Department) as one of several advanced mathematics courses from which computer science majors must take two -- seems to be as strong a requirement as any. And even prestigious, research-oriented institutions (with emphases in applications of logic to computer science) often

only informally "strongly encourage" their students to take logic.

Our students' only standard exposure to logic is in the already overworked course in discrete structures. But there it is almost always trivial truth-table logic and never predicate logic to any depth whatsoever. Yet our majors continue to be required to take a full year (or more) of calculus, a discipline that simply does not permeate computer science as logic does. But, it's argued, calculus exposes the student to a coherent and rigorous study of great power, elegance, and historical value. Are the presentations of logic *today* any less coherent, rigorous, powerful, elegant, historical? No, and, moreover, logic is increasingly important to a full participation of computer scientists in their field. Of course, many -- not all -- of the applications of logic derive from what is now advanced, graduate-level material. But this is all the more reason to introduce the requisite logic background at the undergraduate level.

II. Logic in Computer Science

Logic touches on virtually all areas in computer science. While it's certainly possible for technicians to *use* systems with significant logical bases, their continuing development and evolution of those systems will be frustrated (à la Gries, above) with but poor understanding of logical principles. And the relevance of logic is no longer confined (if it ever was!) to mere truth-table propositional logic for circuit design. Significant strides are now made with techniques from deeper areas of logic such as model theory, proof theory, combinatory logic, and non-classical logics, using such notions as completeness, consistency, axiomatics, natural deduction, Skolem functions, λ-calculus, constructivity, etc.

As an example, consider the applicable area of database systems where deep applications of logic are necessary for a variety of purposes. At the most fundamental level are the various alternatives for knowledge representation and data models (relational, entity-relationship, etc.) and query-processing languages relying on issues of model theory, completeness, consistency, and deduction. And as databases become larger and more complex, isolating the effects of inevitable local inconsistencies from the entire system becomes paramount. Work such as in relevance logics addresses this problem. Current research, relying heavily on logic, includes deductive databases and expert systems; expressiveness; dynamic/temporal modeling and temporal logics (for the dimension of time in databases); knowledge-based systems with incomplete and tentative information requiring modal and fuzzy reasoning; natural language interfaces; and semantics.

Without belaboring the point, brief highlights of the importance of logic to many core areas in computer science include the following necessarily partial and not mutually exclusive listing:

* "theoretical" computer science (of course!)
  automata, formal languages, computability, complexity, recursive function theory;

* artificial intelligence
  deduction systems, expert systems, cognitive science, formalisms, automated proofs, natural language processing;

* programming languages / data structures
  logic programming (PROLOG is but one such language), resolution, functional languages, semantics (Hoare, denotational, procedural, realizability), language design, computational completeness, data abstraction/operations, type theory, object-oriented approaches, parallel processing (optimality and equivalence to sequential algorithms);

* database systems
  discussed above;

* software engineering
  program verification, including testing (path manipulation) and correctness, formal specifications and program design, executable specifications;

* hardware
  circuit design/optimization, hardware design languages, processor verification, correctness of OS kernel, language implementation on given processors;

* philosophical foundation for CS
  profound correspondences between reasoning and computation, formal systems, constructivity as a basis for CS influencing language design, semantics, etc. (computer science as "applied constructivity").

Briefly elaborating a bit of this last area, Boyer and Moore discuss the "formalization problem": how to use mathematics and logic to formalize everyday concepts in CS -- that formulas mean what we want them to mean. Reminiscent of Gries' remark, these authors note that this

> is hard because there is a major philosophical gap between the user's personal intentions and desires and any formal rendering of them. But many users ... wade into this 'specification problem' without fully understanding how treacherous it is. ... Unless you really understand the formal logic you will not be able to capture ... the concepts and relationships you will be grappling with when you apply the logic. [2]

III. Implications for the Curriculum and Another Tirade

It's worth remarking that the partial listing above does not represent obscure corners of computer science of interest to academics only. Rather these are some of the "hottest" domains, concepts, and keywords in contemporary computing. Such topics dominate the prestigious conferences and symposia in computer science today, such as Principles of Database Systems (PODS), Theory of Computing (STOC), Foundations of Computer Science (FOCS), Design and Implementation of Symbolic Computation Systems, etc. In fact, even if the deeper concepts of logic weren't relevant to CS (they are!), much of the notation for communication among computer scientists is logic notation. Indeed, it is easy to find a CS journal or conference proceedings without a single integral or derivative, but virtually impossible to find one without logic terms and notation.

Concepts from calculus are relevant in the above listing of topics only sparsely, if at all (calculus is useful in CS for numerical methods and for certain analyses -- for example, of complexity). So one might justifiably argue for an increasing courseload of discrete topics and a genuine logic sequence to replace calculus. Why, in an era of continuing curricular changes and a very rapidly growing "core" of topics for a CS major, this sort of suggestion is always seen as too radical to merit serious discussion is somewhat puzzling. The theme of this Conference is "From Generation To Generation," and so it seems appropriate to evaluate what from past generations is of the widest relevance for our present curricular needs and constraints. McCarthy stated (1967, quoted in Hoare and Shepherdson), "It is reasonable to hope that the relationship between computation and mathematical logic will be as fruitful in the next century as that between analysis and physics in the last" [7]. There is but small hope, however, unless we finally begin to train our students in the tools of logic.

It's time for our field to enter its adulthood with the tools appropriate to that adulthood (alluding to the coincidence that this is the 21st birthday of SIGCSE! -- those most concerned with CS education). Calculus should be seen as a tool for certain specialized aspects of our field, not as a childhood security blanket, clutched tightly because we've always done so -- the arguments for the necessity of a calculus background for *all* students in the sciences often seems to boil down to something like "it builds character."

Beckman has pointed out that with the computer as a tool, calculus is becoming more experimental in nature and that

> it seems likely that the basic mathematical training given computer science stu-

> dents in the universities will include more work in this mathematics of the discrete with, perhaps, a reduced emphasis on the study of the Calculus ... . [1]

And, more germane to our argument for logic here, he anticipates objections from those not familiar with contemporary treatments of the subject:

> The classical works in this area, written, say, a century or more ago, clearly belonged more to philosophy than to mathematics ... . Very recent books on mathematical logic [after its rigorous development in the early twentieth century] often include extensive treatments of decision and proof methods with detailed coverage of those algorithmic or computational procedures that are significant in these studies. [1: insertion added]

At the very least, the work of Ralston, Prather, and others, supported by the Sloan Foundation a few years ago, might be resurrected. This was an attempt to enhance discrete mathematics and better integrate it into the first-year/sophomore curriculum for CS majors using a "short calculus" course. This goal should receive renewed attention -- not only for the reasons of coping with the "crisis" of a "bloated calculus" and assuring that computer science students receive education in discrete mathematics early on. Indeed, there was motivation for such an approach then, even without the added emphasis on logic presented here.

But the integration would add a tremendous coherence and elegance to calculus and exercise a myriad of fundamental mathematical tools in the process. For example, we would encourage a return to the algorithmic aspects underlying calculus, largely ignored for a hundred years. [As an aside, Prather's approach to the Sloan project [8] was to teach discrete mathematics during the first third and a "short" calculus for the remainder of the year. Thus the first lecture of calculus could be on the uncountability of the reals -- thereby marking the boundary between discrete and continuous mathematics from the outset!].

To pursue this argument, Dijkstra has written that "it helps to view a program as a formula" and that mathematicians have all but ignored the programming challenge because "... programs were so much longer formulae than [mathematics] was used to that it did not even recognize them as such." [4]

Thus, building on ideas of Greenleaf [5] and echoing Dijkstra, we can regard the derivative as a single large formula:

> Formal system: analytic functions f constructed inductively from numerous base functions (e, sin, ln, etc.) by means of basic operations (addition, multiplication, composition, etc.).

```
Specification:  Der(f,x) = f'(x).

Algorithm Der(f,x);
case f(x) of
g(x) + h(x): Der := Der(g,x) + Der(h,x);
g(x)·h(x): Der := h(x)·Der(g,x) + g(x)·Der(h,x);
g(x)/h(x): Der := ...
                  ...
g(h(x)): Der := Der(g,u)·Der(h,x);
          u := h(x);
                  ...
c: Der := 0;
x^r: Der := r·x^(r-1);
ln x: Der := 1/x;
                  ...
sin x: Der := cos x;
end.
```

Here the base functions ground the recursion. Now almost the entire differential calculus may be seen as but a proof of correctness for this algorithm! So, after suitable motivations and definitions, the specification is met by this algorithm (formula). For instance the proof of the chain rule validates the single appropriate case above.

Few, if any, of these concepts are available to students of Calculus I: detailed treatment of function (one of the most fundamental notions in mathematics), mathematical induction, inductive definition, algorithms and correctness, recursion, and proof. In fact, it's often a nuisance that computer scientists, in their already full discrete mathematics course, must take precious time to teach some of these most fundamental tools *of mathematics* to students who didn't learn them in a year course in calculus! If students had this background, calculus would then be a marvelous tying together and application of all these fundamental topics from discrete mathematics! Differential calculus (with its theorems) is just one more important algorithm (with its proof of correctness) among many in computer science.

As a final plea for the incorporation of logic into the CS curriculum, we may cite Dijkstra again:

> A programming language, with its formal syntax and with the proof rules that define its semantics, is a formal system for which program execution provides only a model. It is well-known that formal systems should be dealt with in their own right, and not in terms of a specific model. ... Right from the beginning, ... we stress that the programmer's task is not just to write down a program, but that his main task is to give a formal proof that the program he proposes meets the equally formal functional specification. [4]

The formal language that permeates computer science is the language of predicate logic. It's a very difficult subject for which we should not rely on self-study. We should teach it.

REFERENCES

[1] Beckman, F.S., *Mathematical Foundations of Programming*, The Systems Programming Series, Addison-Wesley Publishing Co. (Reading, MA), 1980.

[2] Boyer, R.S. and J.S. Moore, *A Computational Logic Handbook*, Perspectives in Computing, Vol. 23, Academic Press (Boston), 1988.

[3] Davis, M., "Influences of Mathematical Logic on Computer Science," in R. Herken (ed.), *The Universal Turing Machine: A Half-Century Survey*, Oxford University Press (New York), 1988.

[4] Dijkstra, E.W., "On the Cruelty of Really Teaching Computer Science," SIGCSE *Bulletin*, Vol 21, No 1, February 1989.

[5] Greenleaf, N. "Algorithms and Proofs: Mathematics in the Computing Curriculum," SIGCSE *Bulletin*, Vol 21, No 1, February 1989.

[6] Gries, D., *The Science of Programming*, Springer-Verlag (New York), 1981.

[7] Hoare, C.A.R. and J.C. Shepherdson, eds., *Mathematical Logic and Programming Languages*, International Series in Computer Science, Prentice-Hall, 1985.

[8] Prather, R.E., "A Freshman-Sophomore Curriculum Integrating Discrete and Continuous Mathematics," in A. Ralston (ed.), *Sloan Foundation Discrete Mathematics Programs*, *MAA Notes*, 1989, to appear.