Thinking Through Computation: A Heresy

Stephen Ramsay

October 24, 2007

Any workshop that proposes to discuss alternatives to the theory of computation must be about heresy, and like all those who find themselves on the brink of schism, I must confess to a certain unease. The essential theory of computation (the Church-Turing thesis and its consequent ideas) is so deeply ingrained in the consciousness of anyone who has thought seriously about computing, that one is tempted ask whether any alternatives would still be within the realm of computation? Can we talk about non-Church/Turing models and still be talking about the same subject?

But when I consider why this subject seems so intransigent—why it feels like a keystone, which, once dislodged, brings the whole edifice crashing down—I am led not to the irrefragable nature of the theory of computation, but to an historical insistence on a relationship between computer science and mathematics. The theory of computation isn't static; it's not at all a settled domain. But a theory of computation that did not behave like a mathematical system is quite literally inconceivable within the present paradigm.

It's not difficult to imagine why this might be so. Computer science arises as a field because certain elegant responses to the *Entscheidungsproblem* (most significantly, the Universal Turing Machine and the lambda calculus) were felt to warrant separate consideration. But the demands of computation (understood in the broadest possible sense) are ancient. If it is hard to imagine truly alternative models of computation, it is perhaps because, in our modernity, we can no longer imagine non-mathematical computation. Or rather, we can no longer imagine a computational theory that does not aspire to be mathematics.

Every aspect of modern computer science is shot through with the notion that mathematical purity will get us closer to computational effectiveness. The pantheon of modern programming languages, to take one example, can be easily arrayed along a line stretching from the impure to pure. Is not C++, with its redundancies, its inconsistencies, its side effects, something like an impure language from the standpoint of computer science? And is not Haskell, with its monads, its recursion, its algebraic data types, and its near total lack of side effects a "pure functional" language? The former, in the words of Meilir Page-Jones, "has sacrificed orthogonality and elegance for random expedience." The latter has, in the opinion of some, come closest to the ideal of "executable mathematics." Yet despite the laws of computational Kashrut, C++ is a wildly successful language, while Haskell is the province aficionados, CS researchers, and people who otherwise have the luxury of not having to write working systems for actual people to use.

In saying this, I risk reducing the profound topic at hand to a mere canard: the tension between town and gown that has always set the working, hard-hatted programmer against the ivory towered researcher. But when I consider my own proclivities as a practitioner, I find I am more closely allied with the latter. I will follow any excuse to write in Lisp, and in general, the languages of purity fill me with

a kind of satisfaction I find sorely lacking in the fallen world of Java.

But I cannot deny that there is a deep problem here—a deep problem with all of this. I am not, after all, a computer scientist. I'm an English professor who has managed to gain some fluency with modern programming and mathematics, and for whom the pleasing collision of human art and computational rigidity holds great intellectual fascination. In other words, a very strange sort of character with no hope of converting anyone. It's all too complex, really. Steve and Meurig put it perfectly:

> Notwithstanding its credentials as one of the most significant intellectual developments of the twentieth century, [the theory of computation] is hard to relate to current practice in multimedia, communications, social computing and so on, and is not popular with many—even very able—students.

It's certainly not popular with my very able students, many of whom are extraordinarily well educated in areas at least as abstruse as discrete mathematics. It's not that they aren't intelligent; it's that the disconnect between the theory of computation and the theory of literature are too far from one another conceptually. This, it seems to me, is precisely the spirit of provocation put forth in the invitation to this workshop. "It is hard to relate."

We have tried all kinds of things in the past. We have focused on graphics; we have experimented with visual programming; we have attempted to make the "problem space" look like the "solution space;" we have have played games, pursued highly simplified abstractions, and engaged in all manner of tricks and subterfuges. But in the end (or rather at the bottom) we find the rigid calculus of automata theory, truth-functional logic, and numbers.

What would it look like if we did away with mathematical purity as the gold standard of computation? It's a risky proposition. Manifestos are just that, and "whats" without "hows" descend quickly into futurism. Yet it may be fruitful to consider the theological consequences of heresy in this regard:

1. Instead of privileging correctness, completeness, and closure, our theory would make a virtue of the partial, the mutable, and the contingent. It would, in other words, honor the insistently human way of apprehending the world. The lack of fit between computation and the phenomenal world would not disappear, but computation would cease to distinguish itself from that world by being correct, complete, and closed.

2. Certain interface values would suddenly come to occupy the lowest substrate of computational engagement. We would speak of the "gestural" and the "intuitive" not as honorifics bestowed upon good UIs, but as integral aspects of the theory of computation as such. The result might well be "user friendly" programming languages, but from a theoretical standpoint, the goal would be a human friendly epistemology for describing calculation.

3. Niklaus Wirth's venerable formula (data structures + algorithms = programs) would be replaced with something more easily conformable to what Boole inappropriately called the Laws of Thought: observations + beliefs = epistemologies.

4. The notion of "computational intractable" data would be replaced by the notion of "phenomenally intractable" programming languages. If data could not be processed by a system (because it's "unclean"), we would come to blame the programming language for its inability to model the domain. Instead

> of "munging" data, we would speak of "munging" computable propositions.

I do not know if such reversals exist. My indoctrination has perhaps been too complete to imagine computation as carnival. Yet I do know that Turing's intervention in mathematics was the product of just such a spirit of misrule. It seems an obvious move to us now, but nothing in the history of mathematics is quite as bizarre as the sentence in "On Computable Numbers" that begins "Let us imagine a machine . . . "