The Calculus Formally Known as Pi

Stan Kelly-Bootle, Author

Dominic Behan once asked me in a rare sober moment (for both of us): “What’s the point of knowing something if others don’t know that you know it?”¹ To which I replied with the familiar, “It’s not what you don’t know that matters, it’s what you know that ain’t so.” I was reminded of these dubious epistemological observations while reading Stephen Sparkes’ interview with Steve Ross-Talbot in the March 2006 issue of *ACM Queue*.² In promoting Robin Milner’s pi-calculus as the provably reliable backbone for BPM (business process management), Ross-Talbot eases our fears of the arcane, abstract pi-calculus axiomatics by stressing that the layman/programmer “would never need to see the algorithms...never need to read the literature, unless you were having trouble sleeping at night.”

This is a perfectly valid attitude in the many areas where academic CS creeps up on practical computing. The very notion of abstraction, in the sense of hiding from us the deeper—nay, boring—details, is a computing *sine qua non* resting essentially on our respect for experts in other domains (until proved otherwise). What *really* happens when you drag-and-drop is beyond all reasonable knowing, so don’t ask! The familiar “need-to-know” criterion is, of course, often diluted to at least the “need-to-know-a-bit-about” urge by inherent human curiosity. Thus we can, without drinking too deeply at the Perrierian spring,³ hope to follow the gist when the gurus’ schisms break out, as they are oft wont to do.

Now I’ve always been a bit of a sets maniac, having met a meta-mathematician, my hero Paul Halmos (name drops keep falling...), whose far-from-naive Naive Set Theory set me earnestly exploring, *inter alios*, Russell, Goedel, Turing, Church, Zermelo, Fraenkel, Cohen, and Milner (none of whom I managed to meet close-up). Formal logic, set, and derived systems theory are not the arid, tautological dead-ends that many suppose, although even noted mathematicians have found the axiomatic approach to be a pedantic barrier to achieving real results! The basic concepts of axioms, their completeness and consistency, and what it means for a proposition to be “true,” “false,” and “provable,” have all thrown up their unfair share of shocked surprises. Hilbert’s dream of build-

The hype

OVER THE PI-CALCULUS

ing complete and consistent axioms for arithmetic via “pure” logic and set theory distracted Peano, Frege, Russell and White-

head, von Neumann, and many others until Goedel’s landmark paper of 1931.

Briefly, Goedel proved that any “finitistic” formal axiom system strong enough to support our everyday, check-book-balancing arithmetic is either incomplete or inconsistent. Naturally, incompleteness is easier to live with than inconsistency. In an inconsistent formal system, all propositions can be proved both false and true, so “Hello, Blue-Screen Wall!” or rather, everything crumbles—screens, walls, and all. Incompleteness, on the other hand, simply [sic!] means that not all true propositions are formally provable within the system. This is annoying but unlikely to upset the balancing of your checkbook. The undecidable proposition *G* that Goedel constructed, cunningly and legally, embedded therein the proposition that *G* is not provable. The truth of this is abundantly self-evident if and only if the formal system is consistent (for *G*-false implies *G*-true, the sure sign of inconsistency, whereas *G*-true, meaning *G* not-provable, provides a consistent but incomplete system).

The sneak-easy [sic!] fix is to extend the list of axioms (finitely) so that the previously unprovable truths are now axioms (no proof needed!) or otherwise provable from within the extended system. Alas, Goedel’s incompleteness arguments can be used to construct a fresh, true-but-not-provable candidate *G* within the extended system. And so on. Much progress has been made by dropping the “finitistic” axiom limitations. As early as 1936, for example, Gerhard Gentzen proved the consistency of arithmetic using “transfinite” induction.

Meanwhile, even earlier in the 1920s, the world of formal systems suffered what many consider to be an even greater blow than Goedel’s. This was the infamous L-S (Loewenheim-Skolem) theorem dealing with the models we derive from our axioms, putting meat on the dry bones, as it were. When we first look at a given formal

Continued on page 54

Continued from page 56

system, we must follow its primitives, rules, definitions, and axioms without being misled by extraneous natural-language hints. Thus, a simple geometrical axiom set may offer primitives called *points*, *lines*, *joins*, and *intersections*, with such axioms as, “There exist at least three distinct points,” and “For any two distinct points there is a unique line called the join of the two points,” and so on, with agonizing precision.

You can hardly be blamed for picturing the familiar objects of Euclidean geometry—and no doubt the formalist responsible was motivated by the same concepts—yet formal systems do not really rest on particular interpretations. In fact, there is a real danger if we start assuming properties of the primitives from their names rather than from those properties implied solely by the axioms. (The history of Euclidean and non-Euclidean geometries is a good illustration of such dangers. Certain properties of parallel lines were for many centuries assumed self-evident although not properly provable from the axioms.)

In my example, you can interchange the traditional meanings of point/line and join/intersection without affecting the axioms’ consistency or completeness (or lack thereof). Or you can equally replace point/line with apple/orange! Yet, paradoxically, our formal systems remain symbol-crunching games, of interest only to those who love playing them (yes, they can be fun), unless they can be shown to usefully model some aspects of the real world. The ultimate “real” consistency of a formal system, many believe, is when it can be shown to model a part of “reality” with a trustworthy record of consistency (such as the pebbles/calculi for counting, or the local Euclidean space we inhabit). The L-S theorems disturb this picture somewhat: any formal system permits many more *essentially* different interpretations than the one “intended.”⁴

Now, if you’ve survived the preambles, on to the pi-calculus. As an elegant formal system, it offers few primitives: just names such as P, Q, x, y,...and some operator symbols with which the names are formally manipulated. The excitement comes when the names are interpreted, as intended, as processes and channels, and the symbols are taken to indicate the ways concurrent processes can interact. An essential feature is the definition of “structural congruence,” which lets you *equate* processes that are essentially the same. The completeness is achieved by the definition $!P \rightarrow P \mid !P$, which is interpreted as allowing an infinite number of spawned processes.

If the pi-calculus helps us to program concurrent processes and detect potential kisses-of-death, then it

is welcomed into the CS toolkit. Comparisons can be made with the lambda-calculus, which helped formalize sequential programming methods. To remind you of the potential gap between a formal system and our daily shopping lists, consider the lambda-calculus definitions:

$\text{true} := \lambda x.\lambda y.x$ $\text{false} := \lambda x.\lambda y.y$

These are not the answers Pontius Pilate was seeking (“What is truth?” *John* 18:38). Indeed, they are just clever squiggles with suggestive names intended to mirror some of the technical properties of our everyday binary logic. L-S tells us there are countably many other valid models—but what they are (or could possibly be) neither Loewenheim nor Skolem can say.

Again, one can ask, how much does the programming infantry need to know about these sometimes-soporific calculi? Just enough, one supposes, to call the appropriate library routines at the appropriate times. [That exhausts your quota of “appropriates” for this month.—Ed.]

Where the schisms have emerged is best located with the paper, “Does Better Math Lead to Better Business Processes?”⁵ where Jon Pyke and Roger Whitehead question both the pi-calculus and BPM hype. They make the reasonable point that BPM and its predecessor, WFM (workflow management), are much more complex in the real world than any system based on the pi-calculus or its extensions. Their chief objection is that the pi-calculus is being touted as the “one true road” to BPM. They also query whether BPM itself is *the* brand-new “big thing” or just a rehash of the Workflow Management Reference Model of 1994. Pyke and Whitehead object to the BPMI (BPM Initiative) one-size-fits-all attitude whereby “any product that is not based on pi-calculus cannot, by their [BPMI] definition, offer a true BPM solution.” Let’s hope that a *process* of concurrent reconciliation emerges satisfying both sides’ axioms!

POSTSCRIPT

Bjarne Stroustrup has kindly reminded me of his own contributions to the “code vs. comments” controversy (this column, March 2006). Since C++ comments are syntactically equivalent to a single whitespace character, they must be considered in the plans to extend operator overriding to include the overriding of whitespace (see <http://public.research.att.com/~bs/whitespace98.pdf>). The initial motivation for overriding whitespace, by the way, is to allow mathematicians to write $x\ y$ rather than $x*y$ for multiplication. Ideally, we would want to write x^*y as xy , which matches ancient traditions, but this creates the

problem of distinguishing `xy` as a 2-char identifier from `xy` as the product of two 1-char variables. Bjarne cites the work of a lesser-known Scandinavian, Bjorn Stavtrup, whom I suppose to be a distant, illegitimate offspring of the mysterious N. Bourbaki.

My YACC (Yet Another Comment Compiler) project will benefit from the proposed meta-comment marker `\` to supplement the usual C++ `//` comment convention (`\` means “treat the preceding strings as comments”). ☺

REFERENCES

1. In literary computing you are allowed to drop names as well as digits. Astute readers will notice that I have bowdlerized Dom’s raw brogue where unprintable words of industrial strength are freely interspersed for emphasis.
2. A Conversation with Steve Ross-Talbot. *ACM Queue* 4 (2): 14-23.
3. Formerly Alexander Pope’s “Pierian Spring,” until my advertising contract with Perrier, the well-known fizzy-water-bottling company from Vergèze, France.
4. Becerra, L, Barnes, R. 2005. Evolution of mathematical certainty. *Math Horizons* 13 (September), Mathematical Association of America; [http:// www.maa.org/math-](http://www.maa.org/math-)

horizons. A splendid but pessimistic summary from Thales to Cook via Leibniz, Kant, Turing, Cohen, et al.

5. Pyke, J., Whitehead, R. 2003. Does better math lead to better business processes? http://www.wfmc.org/standards/docs/better_maths_better_processes.pdf.

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

STAN KELLY-BOOTLE (<http://www.feniks.com/skb/>; <http://www.sarcheck.com>), born in Liverpool, England, read pure mathematics at Cambridge in the 1950s before tackling the impurities of computer science on the pioneering EDSAC I. His many books include *The Devil’s DP Dictionary* (McGraw-Hill, 1981), *Understanding Unix* (Sybex, 1994), and the recent e-book *Best of Computer Language: A Stan Kelly-Bootle Reader* (CMP Media, 2005). *Software Development Magazine* has named him as the first recipient of the new annual Stan Kelly-Bootle ElecTech Award for his “lifetime achievements in technology and letters.” Neither Nobel nor Turing achieved such prized eponymous recognition. Under his nom-de-folk, Stan Kelly, he has enjoyed a parallel career as a singer and songwriter.

© 2006 ACM 1542-7730/06/0500 \$5.00