

The Meeting

Peter Grogono

Some years ago, I was asked to chair a workshop with a mandate to evaluate recent contributions to theoretical issues in computer programming. My protests that this was not within my area of expertise were quickly dismissed with the observation that I had been selected for my skills as a peacemaker. When I arrived on the first morning, the participants were discussing our venue, a splendid example of baroque architecture in southern Germany.

“Who on earth chose this decrepit pile of rubble?” Tom was complaining. Although he ranked near the top in his chosen field, algebraic topology, Tom spent much of his time unemployed. It was hard to say whether this was due to a surplus of algebraic topologists or his caustic personality.

“In the village, they call it the ‘house of a thousand passages.’” Isobel was unmarried, and frequently claimed that this was because she had never encountered a man who met her standards. She had held a chair in applied mathematics at a prestigious university since the age of 29, making the claim at least plausible.

“Gross underestimate! I’ve been lost more often than that already,” Harry was the only programmer in the room. The others called him “Harry the Hacker”.

“It’s not nearly as bad as one of your flowcharts,” retorted Tom.

Leo was a heavily bearded expert in formal logic who smoked a pipe almost continuously. When he wasn’t smoking it, he was cleaning it in various ways, to the unanimous disapproval of the group. “The real problem is not so much the number of passages,” he announced, pulling a dangerous looking implement from his pocket, “as the fact that none of them seem to go anywhere.”

“Yes, they do. Around in circles,” responded Tom.

“They certainly don’t lead to the bathrooms.” Michael was also a mathematician, but I never discovered his area of expertise. When I’d asked him what he liked doing, he’d replied “Random embeddings of locally smooth manifolds,” which didn’t help much.

“There are bathrooms in this place? I thought that we were supposed to—” Harry trailed off as Isobel glared at him.

I announced the first session, during which Dr. Robert Revett, who had recently received his Ph.D. was to present a paper based on his dissertation.

“This paper introduces—” he began.

“It does no such thing,” interjected Paul, a gangling mathematician who devoted a generous portion of his considerable energy to preventing us from making statements that did not conform to his own standards of exactness. “*You* do the introducing, please, not the paper.”

“In this paper, I introduce—,” Bob hastily revised.

“And it’s not a paper, it’s a talk,” said Paul.

“I’m sorry, I’ll start again. In this talk, I introduce a new language called—”.

This announcement was greeted with a chorus of groans from around the table.

“Not *another* language. Don’t you guys have enough already?” Michael asked, looking at Harry.

“Welcome to Babylon.” Harry’s voice was resigned.

“Order!” I called. “Please let Dr. Revett continue.”

“In this talk, I will introduce a language called Gödel.”

“Why on earth would you call a language *girdle*?” asked Harry. “Is there a subtle pun on ladies’ foundation garments?” He glanced quickly at Isobel, and went on, hastily, “Or do you want to girdle the world?”

“Gödel is spelled gee oh umlaut dee ee ell,” replied Bob. “Gödel was a German mathematician—”

“Czech logician,” interrupted Leo.

“Moravian,” said Isobel.

“German,” said Tom. “And Jewish.”

“So what?” asked Leo.

“What do you mean, ‘so what?’” responded Tom.

“Why does it matter whether he was Jewish?”

“Kurt Gödel was born of German parents at Brünn, Moravia, in April, 1906,” said Paul, patiently. “Brünn is now Brno in what was until recently Czechoslovakia. Gödel was not, to the best of my knowledge, Jewish. You can call him a mathematician if you like but, since his primary contributions were in the field of logic, calling him a logician is more precise if not more accurate.”

“—who made important contributions to logic,” continued Bob. “In naming my language after him, I am following the tradition, established by Niklaus Wirth, of naming programming languages after mathematicians. Pascal, for example.”

“Euler.” “Euclid.” “Haskell.” “Turing”, came the chorus from around the table.

“Ada,” said Michael.

“Not a mathematician,” was the instant rejoinder from Leo.

“Why not? Because she was a woman?” asked Isobel, quietly.

“Who was Modula?” asked Harry. “Wirth designed that. And what about Miranda? Who was she?”

Leo’s deep voice drowned out the babble.

“Prospero’s daughter in *The Tempest*. And Oberon is a character in *Midsummer Night’s Dream*. It seems that the supply of mathematicians is exhausted and we are engaged in a succession of Shakespeare’s *dramatis personæ*.” He continued. “Gödel’s incompleteness theorem states that, within a formal system *ess* with at least the power of simple arithmetic, it is not possible to prove completeness. Gödel proved this by the novel step of encoding the system within itself, so that statements *about* *ess* could be expressed *within* *ess*. He demonstrated the existence of a logical sentence *gee* which says, and here I anthropomorphize slightly, ‘I am unprovable.’ By its construction, *gee* cannot be proved, and so it is clearly true. Since it is true, the system contains an unprovable but truthful sentence and is therefore incomplete.”

“But you can’t just say ‘it’s clearly true’—if you’re a mathematician, you’ve gotta prove it,” objected Harry.

“Yes, you can prove it formally, but only in a system that is more powerful than the system in which you started. In the more powerful system, gee is a theorem,” answered Leo.

“Ah, but the more powerful system, let’s call it ess prime, in which you can prove gee, must, by Gödel’s theorem, contain a sentence gee prime, which is not provable in ess prime,” said Tom.

“Of course. But gee prime can be proved in a yet more powerful system ess prime prime, which naturally contains an unprovable sentence gee prime prime,” replied Leo.

It seemed to me that this kind of discussion might go on all afternoon, so I held up the palm of my hand towards Leo, who was making claims about something he called “gee omega”, and signalled Bob to continue.

“The language has been designed to facilitate correctness proofs of programs, and—,” started Bob.

“Precisely what do you mean by ‘correctness proof?’” asked Paul.

“I mean that, given a program pee and a specification ess, I can prove that pee implements ess correctly,” replied Bob. “To put it less formally, I prove that pee does exactly what ess says it should do.”

“Yes, but your proof is, at least in principle, a *formal* proof, I hope?” asked Leo. “In other words, you can express all this in a formal system?”

“Of course. Any program written in Gödel can be reduced to a sentence in a formal system. The specification is another sentence in the same formal system. I prove the equivalence of the two sentences,” replied Bob.

“Can one of you mathematical types tell me what a formal system is in words of seven syllables or less?” asked Harry. There was a momentary silence while everyone considered this.

“Consult any introductory text on logic,” offered Isobel.

“A formal system has syntax but no semantics,” suggested Michael. “Form but no content.”

“A formal system has a fixed set of axioms and rules,” said Paul, in the weary tone of a professor addressing a first-year undergraduate class for the fiftieth consecutive year. “It is ‘formal’ in the sense that we are not obliged to ascribe meaning to the rules.”

“Yes, but we can have an interpretation which *does* ascribe a meaning to the rules,” objected Leo.

“Yes,” continued Paul, “one way to demonstrate the consistency of the system—in other words, its freedom from internal contradiction—is to provide a model for it. Informally, the model gives the system a meaning.”

“Do you have an interpretation for Gödel?” Isobel asked Bob.

“Of course. Every Gödel program has an interpretation in the second order polymorphic typed lambda calculus,” responded Bob. “It’s quite standard in that sense.”

“So pot luck,” Harry murmured.

“I beg your pardon?” said Michael.

“My acronym for *Second Order Polymorphic Typed Lambda Calculus*—so pot luck,” replied Harry.

“But isn’t the second order poly whatever you said a formal system itself?” I asked.

“Yes,” replied Bob.

“So, being formal, it doesn’t really have a meaning. Therefore, you are giving your programs a meaning by mapping them into a system that doesn’t have a meaning,” said Harry, settling back into his chair with the air of a contented cat.

“Not really. There are well-known models of second order lambda that are based on Scott domains,” said Bob.

“And a Scott domain is—?” I queried.

“A complete lattice, downward closed under continuity, often with the Egli-Milner ordering but other orderings are possible and occasionally preferable. The ordering ensures that continuous functions respect monotonicity,” said Paul.

“Quite so,” said Michael.

“Sorry I asked,” I mumbled.

“Ah, and I had always thought it was some kind of penal institution for demented Celts,” said Harry.

Jock was the only electrical engineer at the meeting. He had, until this point, remained silent, seemingly preoccupied with the elaborate scrollwork on the ceiling. His name was actually Andrew, but everyone called him “Jock” because he had a mild Scottish accent. In response to Harry’s supposition, he brought his gaze down to the level of the room, growled “Take care, laddie,” and moved his attention to the chandelier.

“Sorry, Jock,” said Harry. “I forgot that you were a demented Celt.”

It might have been interesting to find out more about Scott domains but, since Paul appeared to be on the verge of explaining them at length, I hastened to bring the discussion back on course.

“Dr. Revett,” I said. “You have claimed that you can prove a program correct with respect to a specification. Can you give us definitions of specification, program, and correctness?”

“Ah, that’s a real can of worms!” Tom sat upright in his chair and looked eagerly attentive.

“A specification defines a class of computations. A specification is not required to be executable but a program *is* required to be executable,” said Bob carefully, and moved to the blackboard. On it he wrote:

$$\{ p \mid p \text{ is prime} \} \cap \{ e \mid e \text{ is even} \}.$$

“This specification,” he continued, pointing to what he had just written, “defines the integer, or integers, that belong to both the set *pee* of primes—two, three, five, seven, and so on—and to the set *ee* of even integers—two, four, six, and so on. I call it ‘non-executable’ because a naive processor would first attempt to compute the set of primes, taking an infinite time, and would then attempt to compute the set of evens, taking another infinite time and, finally, would compute their intersection.”

“Your concept of computation is not just naive, it’s utterly stupid,” said Tom. “The intersection of those sets is the singleton set containing two, as is immediately obvious to the most meagre intelligence and probably even to a computer.”

“Compiler,” murmured Harry.

“Precisely. Here is the program corresponding to the specification,” said Bob, and wrote

on the blackboard underneath the specification. “I am not surprised that it is obvious to *your* meagre intelligence, Tom, but, nonetheless, most modern compilers could not deal with it, Harry. Or, of course, more complicated specifications of the same kind.”

“Well, I can imagine a programming language in which your spec would be quite acceptable as a program, and I could write a compiler that would figure out a fast way of computing the result,” said Harry.

“So a specification is really just a program in a higher level language,” said Isobel. “I would like to make the converse observation. By your definition, a program is really a specification, because all but the most trivial computations can be performed in more than one way.”

Michael moved to the blackboard and solemnly wrote

$$2 \times N$$

The murmuring stopped as if a significant event was about to happen. Looking around the table, I realized that even the mathematicians could agree that multiplication by two was a topic that deserved serious attention. After a moment or two during which we all contemplated the ineffable, Michael continued brusquely.

“I could compute two times *enn* in a number of ways. I could use a multiply operation if my processor was equipped with such. I could replace the multiplication by an addition, *enn* plus *enn*, which is equivalent. Or, assuming two’s complement binary representation, I could perform an arithmetic left shift. So I put it to you, gentlemen—and lady,” he added hastily, with a guilty glance at Isobel, “that two times *enn* represents a class of computations and is therefore, by your own definition, Dr. Revett, a specification.”

“Exactly as I said,” murmured Isobel.

With a flourish, Michael threw the chalk on the floor, perhaps confusing it with a gauntlet, and returned to his seat. Since no one seemed ready to respond to the challenge implied by the chalk, I nodded to Bob.

“Naturally, the construction of the correctness proof is carried out by a program.”

“Aha, the issue of completeness raises its ugly head once more,” cried Leo. “Your program prover proves the correctness of programs. But it is *itself* a program. Can it prove its own correctness?”

“No,” replied Bob. “That’s a well-known result.”

“It’s hardly well known, but I’m sure someone’s proved it,” said Paul.

“But,” continued Bob, “suppose we call the program prover *pee pee*. Then there is *another* program *pee pee prime* that can prove the correctness of *pee pee*.”

“Ha! But not its *own* correctness, eh?” shouted Tom. “For that, you need *pee pee prime prime*. And then you need *pee pee prime prime prime* to prove *that*.”

This seemed to be another issue that might go on for a while. Before Leo started burbling about “*pee pee omega*”, I called the meeting to order again.

“This discussion is having an unfortunate effect on me. Excuse me for a moment,” said Harry, and left the room.

“Okay, Bob, let’s say you’ve got your correct program,” I said. “Can you convince us that it will give the right answer when you run it on a computer?”

“Well,” replied Bob. “There are several issues. First, since the computer cannot execute Gödel programs directly, I translate them into C and compile that into machine language.”

“Have you formalized the semantics of C?” asked Leo.

“You haven’t had enough formal systems yet?” Michael asked him.

“In principle, a programming language is a formal system which can be axiomatised,” replied Bob. “The translation to machine code is a syntactic operation, and the correctness of the translation process can then be formally proved. Even a processor is a kind of model of a formal system. In practice, however, the variety and complexity of contemporary machine architectures and, in particular, concurrency phenomena such as pipeline stalling, make it extremely tedious and, to a large extent, pointless to formalize either the machine language or the process of translation.”

“In other words,” said Tom, “you couldn’t be bothered.”

Since I felt that it was time to establish more definite goals for the discussion, I asked Paul for a review of the ground that he felt we had covered.

“We have a specification, *ess*,” he began, “which might actually be a program”—he nodded at Harry—“and a program, *pee*, which might actually be a specification.” He nodded at Isobel. “We can express the meaning of *ess* and *pee* in a system that, being formal, has no meaning, and we can prove, using a program *pee pee* that is in principle incomplete, that these meanings that are not *quite* meanings are, in some abstract mathematical universe, equivalent, at least up to isomorphism. Moreover, we can use an unproven translator to compile *pee* into another program, or possibly specification, written in some undefined machine language, that performs the computation on a processor that we do not bother to describe in a formal way. And Bob claims that the result of all this will be, in some sense that he has not yet adequately defined, correct.”

“Absolutely,” cried Bob. “And I can do more—”

“But suppose some clumsy mathematician,” Leo began, with a meaningful look at Tom, “should trip over the power cord, thereby prematurely terminating the computation. What then?” He made a gesture apparently intended to indicate the collapse of a rococo palace of cards.

“My theory does not account for clumsy mathematicians. Or the computer being struck by lightning. Or—”

“Or two cosmic rays zapping the same memory chip simultaneously,” said Tom, helpfully.

“Or the computer being engulfed in a pungent cloud of toxic goop released by Leo’s pipe,” said Michael.

“Or a herd of gnus pursued by wildebeest stampeding through the machine room,” said Harry, happily joining the fray as he returned to the room.

“Wildebeest do not pursue gnus,” said Paul. “They are in fact the same animal. ‘Wildebeest’ and ‘gnu’ are variant names for the species *Gorgon taurinus*, a member of the antelope family that lives in Africa and subsists primarily on grass and leaves. Furthermore, ‘wildebeest’ is a Dutch word that should be pronounced, approximately, ‘vill de baste’ but is usually pronounced ‘will de beast’ by literate speakers of the English language. It is *not* pronounced ‘wild beast’.”

“Okay then,” responded Harry, “a herd of *gorgones taurini* pursued by carnivorous logicians stampeding through the machine room.”

I felt that we were drifting away from the topic again. “Order! Order!” I cried. “Let’s agree that a physical phenomenon might disrupt the symbolic processing. Bob?”

Bob shrugged. "I have made the standard assumptions," he said. "I am concerned with formal techniques for the derivation of correct programs. If you want to run the programs on a boxful of unreliable junk, that's your problem."

This comment alerted Jock, who stopped admiring the elaborate candelabra that surmounted the mantelpiece and rose slowly to his full height of about five and a half feet. He extended a quivering index finger in the general direction of Bob.

"'Boxful of unreliable junk'," he quoted scornfully. "I would like ye to fully appreciate, *Doctor Rivet*, that what ye call a 'boxful of unreliable junk' is the only thing around here that ever gets anything done. *You* talk about the 'software crisis'. *We* are engineers. *We* don't talk about the 'hardware crisis'. We don't piss around with formal systems that don't mean anything. We don't natter on about gee-gees and pee-pees. We write specs and we build machines that meet those specs and the machines bloody well *work*. If they didn't work ye'd nay be here because there'd be naught for ye to talk about. Ten years ago I swore I'd never come to another of your damned software meetings and I should'a stuck wi'that."

He picked up his file of papers from the table, dropped it, bent to pick it up, knocked over a chair, and strode to the double doors. After pushing the left door, pulling the right door, and pushing the right door, he finally succeeded in escaping by pulling the left door, although not before he had dropped his file again. There was a silence, eventually broken by Leo.

"When a kid gets a soldering-iron, he immediately loses all sense of values," he said, sympathetically.

With admirable presence of mind, Bob placed his first transparency on the overhead projector. It consisted mostly of dense formulas containing a variety of symbols that I had never seen before. Everyone's attention moved to the screen and the meeting proceeded in a surprisingly orderly manner, as the mathematicians took turns explaining, in their individual ways, why the equations could not possibly establish Dr. Revett's claims. I congratulated myself, privately, on my peacekeeping abilities.