"Real mathematicians don't prove"

Roughly speaking, there are two ways in which people try to reason about programs; I shall distinguish them as "the postulational method" and "the operational method".

The first method is called "postulational" because it postulates how the program text and the specification define the lemmata to be proved in order to show that the program meets the specification. (In effectively discharging this proof obligation, the predicate calculus has shown itself to be an indispensable tool.) The postulational method treats the program text as a mathematical object in its own right, i.e. semantic equivalence of two programs means that they meet the same specifications.

The second method is called "operational" because it tries to analyse what computations could be evoked under control of the program and to establish that each possible computation is compatible with the given specification. It relies on a computational model with respect to which the program text is interpreted as executable code.

The tragedy of today's world of programming is that, to the extent that it reasons about programs at all, it does so almost exclusively operationally. I call this a tragedy because, from a purely technical point of view, the operational method is by several orders of magnitude inferior to the postulational one. With growing size or sophistication of the program, the operational argument quickly becomes impossible to carry through, and the general adherence to operational reasoning has to be considered one of the main causes of the persistence of the software crisis.

A possibly very fundamental flaw of the operational method is that it begs the question how to reason about algorithms, because it translates the possible effects of a given algorithm into those of another one, viz. the program interpreter (i.e. the abstract machine that underlies the computational model). We shall not pursue that potentially severe shortcoming here. Our current concerns are much more pragmatic: by admitting - or should we say: by generating? - the possible computations into our considerations we open the door to a combinatorial explosion, the effect of which quickly defies exhaustive analysis. [Remember the archetypical programmer's excuse for a bug "Oh, but that was a very special

case."!] Instead of finding out how to cope with the effects of such combinatorial explosions, it is much more effective to prevent the combinatorial explosion from occurring in the first place; this is what the postulational method achieves by not taking into account that the program text admits the interpretation of executable code. The postulational method deals with the program text as a parsed but otherwise uninterpreted formula.

As soon as the postulational method began to be forcefully advocated, it met equally forceful opposition, all of which was quite predictable. I mention, by way of illustration

(0) It is of a "theoretical level" that "places it beyond the scope of most amateurs", or: "but that would require a lot of education". (Standard answers varying from "I never claimed that amateurs should be able to program well." to "Well, education is my business.".)

(1) The postulational method is of no relevance for the real world, for real programmers don't think that way. (Standard counterquestion: "Do you mean to say that I am a virtual programmer?".)

(2) It cannot be any good because backward reasoning and weakest preconditions are counter-

intuitive. (Standard answer: "If a simple calculus can achieve what is so "counter intuitive" that it is beyond the unaided mind, so much the better for that simple calculus.".)

(3) It may be okay for toy programs, but you'll never be able to apply it in the case of real programs. (Standard answer: "Yes, scaling up is a problem, but the operational argument becomes much sooner impossible than the postulational one.".)

(4) By imposing such strict logical constraints, you stifle the programmer's creativity. (Various answers are possible, such as "Unbridled creativity has done more harm than good." and "If the programmer really wants to be an impressionistic poet, he is in the wrong business.". For a more sophisticated audience you can explain that, at each stage of the design, the explicit statement of the designer's obligations is at the same time an explicit statement of his freedom, thereby inviting him to explore alternative designs the traditional programmer almost certainly overlooks. If time permits, you can give an example.)

(5) Etc.

The moral of the story is clear: real programmers don't reason about their programs, for reasoning isn't macho. They rather get their substitute for

intellectual satisfaction from not quite understanding what they are doing in their daring irresponsibility and from the subsequent excitement of chasing the bugs they should not have introduced in the first place.

* * *

The above battles are only too familiar to anyone with even the faintest acquaintance with programming methodology. The reason why I described them is that they are the battles that pervade the rest of mathematics: the "formalists" that try to do mathematics by manipulating uninterpreted formulae accordingly to explicitly stated rules and the "informalists" — only they don't call themselves by that negative name: presumably they present themselves as "the real mathematicians" — who constantly interpret their formulae and "reason" in terms of the model underlying that interpretation.

After having given examples of "pragmatic demonstration of the creative power of the postulational method", E.T. Bell continues with "Mathematicians and scientists of the conservative persuasion may feel that a science constrained by an explicitly formulated set of assumptions has lost some of its freedom and is almost dead. Experience shows that the only loss is the denial of the privilege of making avoidable mistakes in reasoning. As is

perhaps but humanly natural, each new encroachment of the postulational method is vigorously resisted by some as an invasion of hallowed tradition. Objection to the method is neither more nor less than objection to mathematics." That last sentence is very clear as to what E.T. Bell thinks worthy of the name "mathematics". He does not classify himself as one of "the mathematicians of a conservative persuasion".

Let us quote some of those by way of contrast: Philip J. Davis and Reuben Hersch. They refer to "the error of identifying mathematics itself (what real mathematicians really do in real life) with its model or representation in metamathematics, or, if you prefer, first-order logic.". In the next paragraph - and now it is beginning to sound very familiar - they dogmatically state "that such [formal] derivations [in first-order logic] are purely hypothetical activities (except for "toy" problems that might be played with in a course in logic)". In the next sentence they refer to "real mathematics, with proofs which are established by "consensus of the qualified." A real proof is not checkable by a machine....". Whereas Bell (in 1940) refers to mathematics as a goal, Davis and Hersch (1981) refer to the average status quo, including all its medieval or older relics.

My trouble with Reuben and Hersch is that their status quo seems so close to their goal. They admit that "a teacher <u>can</u> teach mathematics and a reseacher can write papers without paying attention to the problem of intuition" but only do so under the proviso "perhaps it would be foolish and self-defeating". My tentative conclusion is
(i) that they have never tried to teach mathematics formally and explicitly without appeal to intuition – if they had, it would have been a most refreshing experience for them and for those of their students that were sufficiently well-educated to appreciate simplification
(ii) that they have never taken the trouble to learn how to let the rules of the formal game guide their "writing of papers" – if they had, they would have discovered how to do mathematics way beyond their former powers.
(Speaking from experience. I can, of course, not expect to convince anybody.)

The consensus model of mathematics seduced A.J. Perlis et al. to argue the futility of the aim of designing programs proven to meet their specification. What they missed is that computing science is probably one of the least medieval branches of mathematics – not of "real mathematics" of course! – .

* * *

For a sociologist of science there is the question why, in contrast to other sciences, in mathematics - I mean "real mathematics as really done by real mathematicians in real life" - the medieval characteristics are still so overwhelming. (E.g. how to do mathematics is not taught explicitly but only by osmosis, as in the tradition of the guilds. The usual excuse - even from otherwise well-reputed professors of mathematics - is that there is no other way.)

I am insufficiently knowledgeable to offer a convincing explanation. I have understood that the Renaissance was a traumatic experience for the intellectual establishment of the day. (Vide the fate of Galileo Galilei.) The notion of science (as understood today) was born during the Renaissance (and grew up during the Enlightenment), and the intellectual establishment of the Middle Ages — i.e. theology and mathematics — had to discover how to cope with it. It took theology - at least the popes of Rome - almost four centuries to solve their problem by divorcing their religious teaching from scientific pretension. It is quite possible that mathematics is even more between a rock and a hard place. By its mere utility its divorce from science was not encouraged. (We should note that, by focussing their attention to provability, the formalists have turned away

from the fuzzy metaphysical notion of "truth". This was quite an achievement: we should remember that even as late as in the previous century mathematics still suffered from such a heavy philosophical pollution that no one less than C.F. Gauss thought it wise not to publish his discovery of non-Euclidean geometry and left it to Bolyai and Lobachewski to receive the flack.)

While, in the Renaissance, the other sciences were just beginning or could start over again with a tabula rasa, mathematics was punished with the burden of many centuries of premedieval mathematics that, in a way, maintained its relevance. It is quite understandable that, at the time, it clung to Euclid.

But this historical explanation is no excuse for the fact that today Euclidean geometry, with all its known defects, is still taught as the prototype of a strictly deductive system. It is not. Its reliance on pictures is quoted as evidence for the value of "geometric intuition", while in fact it is only a symptom of the defects of Euclid's axiomatization. (In passing I would like to add that I see more and more evidence that it is precisely the informality of the pictorial argument that defies the development of heuristics by means of

which such arguments can be designed in an orderly fashion. This is in sharp contrast with the highly effective heuristics that could be developed for strictly formal environments.)

The pretence with which Euclidean geometry is taught today (viz. that it is a strictly deductive system) is a great, big lie. Its general acceptance (and the wide-spread praise of "intuition" in its wake) is telling: mathematics today is still a discipline with a sizeable pre-scientific component, in which the spirit of the Middle Ages is allowed to linger on.

Eventually completed at Austin
on 24 Januari 1988

prof. dr. Edsger W. Dijkstra
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188
USA