Introducing a course on calculi

The functions I grew up with, such as the sine, the cosine, the square root, and the logarithm were almost exclusively real functions of a real argument. Sometimes, but not very explicitly, the argument could be allowed to be a little bit more complicated, such as the maximum of two values, a real function defined on a pair of real values, or on two values: whether the maximum was a function of one argument (which had to be a pair) or was a function of two arguments (each of them a real number) was a question that was avoided. Didn't we talk about the types of the function arguments, we did not talk about the types of the function values either: we did not need to, for to all intents and purposes, all our functions were of type real. (Later this was extended to the type complex, but that was about it.) The nett effect was that I was extremely ill-equipped to appreciate functional programming when I encountered it: I was, for instance, totally baffled by the shocking suggestion that the value of a function could be another function.

Outstanding features of functional programming are (i) the rich type structure needed to make the paradigm of function abstraction and function application sufficiently flexible, and (ii) the predominance of functional composition as structuring device. The clarity and economy of expression that the language of functional programming permits is often very impressive, and, but for human inertia, functional programming can be expected to have a brilliant future, the more so because today's computers admit quite efficient implementations of functional programming languages.

But this is not the complete story. You see, there are people —and I am one of them— that think they have good reason for not wanting to delegate all calculation to our machines, because they see a rôle for unprogrammed calculations done by themselves— for instance in the derivation of proofs and programs—. Whether certain operations are convenient to carry out or not, of course greatly depends on the tools one has to do it with, and in this context we have to accept that a random-access

store and the combination of pen and paper are two storage media with totally different characteristics. In particular, evaluating a functional program with pen and paper is a pain in the neck! In contrast, this course focusses our attention on calculi geared to the use of pen and paper (a medium that has more virtues besides being traditional).

I expect most of our time to be spent, first on the predicate calculus, and then on the relation calculus. For the sake of greater utility, we shall immediately introduce a generalized predicate calculus which later will be used as the framework [sic!] in which to define the relation calculus.

The traditional boolean type is two-valued: {true, false}. If so desired we can model these two values by the 2 subsets of a 1-element universe, the traditional association being that of true with that universe and of false with the empty subset. Our generalization consists of a (still boolean) type whose values can -but don't need to- be modelled by the subsets of a larger universe.

As a result, this calculus is widely applicable; for those familiar with it, it often becomes the tool of almost daily reasoning.

The reason to deal with the relation calculus is two-fold. Firstly, relations are a mathematically succinct way of capturing the semantics of not necessarily deterministic programs, secondly, because the relation calculus is more structured than just the predicate calculus, it provides an ideal terrain for the development of heuristics for proof design.

* * *

I think it wise, and only honest, to warn you that my goal is immodest. It is not my purpose to "transfer knowledge" to you that, subsequently, you can forget again. My purpose is no less than to effectuate in each of you a noticeable, irreversable change. I want you to see and absorb calculational arguments so effective that you will never be able to forget that exposure. I want you to gain, for the rest of your lives, the insight that beautiful proofs are not "found" by trial and error but are the result of

a consciously applied design discipline. I want to inspire you to raise your quality standards. I mean, if 10 years from now, when you are doing something quick and dirty, you suddenly visualize that I am looking over your shoulders and say to yourself "Dijkstra would not have liked this", well, that would be enough immortality for me.

Your obligation is that of active participation. You should not act as knowledge-absorbing sponges, but as whetstones on which we can all sharpen our wits. If you don't understand me, ask for clarification; if I am going to fast, slow me down. (If I am going to slow, you may try to speed me up, e.g. by yawning?) finally, don't expect me to motivate you, but during "office hours" I would like to get personally acquainted with you.

Austin, 30 August 1995

prof. dr. Edsger W. Dijkstra
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78750-8138
USA