"Predicate Calculus and Program Semantics", fall 1989

Our position is -and has now been for a long time- that it does not suffice for a programmer to deliver a program text with the message "here is my program and I hope that people will like it," in very much the same way as a composer delivers his score with the message "here is my symphony and I hope people will like it". There is more to programming.

It does not suffice either if the programmer just says "here is my program and that is what it can do for you" if the latter claim has no convincing support and is therefore no more than a hope.

It is our position that a solid product consists of a triple: a program, a functional specification, and a proof that the program meets the functional specification. Design disciplines for such triples exist, but are <u>not</u> the topic of this course. (In the case of imperative programs, it is the topic of one of my other courses.) Such proofs are based on laws about the programming language in question. This course is about such laws in the special case of imperative programming; for the sake of clarity we shall confine ourselves to what seem about the simplest programming languages that are still interesting.

Using "Hoare triples" to pose them, examples of such laws would be the answer to questions such as

"Is $$\{P\}\ S\ \{Q \wedge R\}$$

the same as

$$\{P\}\ S\ \{Q\} \ \wedge\ \{P\}\ S\ \{R\} \quad ?$$

And, if so, does the same hold for conjunction's generalization the universal quantification, i.e. is — with dummy x over some range —

$$\{P\}\ S\ \{(\forall x ::\ Q.x)\}$$

the same as $$(\forall x ::\ \{P\}\ S\ \{Q.x\}) \quad ?"$$

Because, in reasoning about programs, we rely on such laws all the time, one of our purposes is to put them on a firm footing. (For instance, the laws that Hoare originally postulated as inference rules will either be subsumed in the predicate calculus, or appear as theorems in our theory.)

If past experience is any guidance, the theory and the way in which it is developed will be most unfamiliar to most of you. It has, however, a major advantage over all alternatives I am aware of: it is so simple that almost all of its proofs need not be "discovered" by a spark of genius or a flash of insight but can be designed quite systematically.

All this convenience could only be reached by adhering all through the theory to a strict calculational proof format, and a major secondary purpose of this course has become to familiarize the student with the benefits that can be derived from a proper formalization, rendered in a notation that is geared to our manipulative needs. Wherever possible I shall draw attention to the conceptual and notational choices that made these simplifications possible. The core challenge of computing science has been summarized as "how not to make a mess of it"; it could very well share this challenge with formal mathematics.

Austin, 4 September 1989

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