Introducing an course on the design and use of calculi

My reason to give this course is my conviction that power and crispness of well-designed calculi is, in general, underestimated, and that in the science and practice of programming, calculi have an increasingly important role to play. How to design a calculus so as to make it geared to our manipulative needs, and how to design for a calculus a strategy for its effective use thus become significant methodological issues.

We shall illustrate what we have understood of these issues by discussing what can be viewed as a family of calculi: the classical predicate calculus, generalized to a boolean algebra, the relational calculus and the regularity calculus (i.e. the calculus of regular expressions). There is a very personal justification for this choice of illustrative material: my own familiarity with and intellectual investment in these calculi; but fortunately, there is a much more objective justification. The predicate calculus is so general that, of course, we can encounter it all over the place: there is (Naur/Floyd/Hoare/Dijkstra/Gries/Owicki) a long tradition of reasoning about programs, sequential or not, in terms of predicates on the state space, which also (Hoare/Dijkstra & Scholten/Morgan) form the basis for theories about semantics and program refinement. The relational calculus and the regularity calculus are a little bit more specific, but still very frequently relevant: for the regularity calculus this is illustrated by the close link between regular expressions and finite-state automata, for the relational calculus by its links with demonic nondeterminacy (Rutger M.Dijkstra/Nguyen), with logic programming (Cousot & Cousot) and with functional programming/type theory (Backhouse). In short, for computing scientists, these calculi are of interest in their own right.

These calculi are so general and ubiquitously useful because they can be defined by such a modest number of --pleasant!-- postulates. The modest number of postulates, moreover, makes them most suitable for illustrative material. This suitability is enhanced by the fact that the little bit of underlying lattice theory that will be used all the time is extremely elegant. It is, in a way, perhaps surprising that in all their simplicity, these calculi are "rich" enough so that the design of effective calculations poses a strategic challenge.

We shall try to rely on as few definitions as possible and to keep our notational conventions as homogeneous as possible. Which concepts to define, and which notational conventions to adopt, should, however, be questions of explicit concern. (This is obvious: since the availability of an adequate notation can make a difference as between night and day, we had better be able to judge the adequacy of proposed or considered notations. Also, in each theory, superfluous distinctions, concepts and theorems are a real encumbrance.)

We shall try to make our calculational proofs as short as possible --without committing the sin of omission!-- and hope to come very close to our aim of designing almost all our proofs without pulling a single rabbit out of the hat.

Austin, 26 August 1992

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