Edsger W. Dijkstra
6602 Robbie Creek Cove
Austin, TX 78750-8138

to Professor Don Batory
Department of Computer Sciences, UT

Austin, Thursday 30 August 2001

Dear Don,

Thank you for your e-mail message of yesterday. Sorry that I did not answer your question immediately, but I wanted to give the "official" answer in the "official" terminology and notation. I found what I was looking for in "Unifying Theories of Programming" by C.A.R. Hoare & He Jifeng, Prentice Hall Europe, (1998).

<u>Notational Preliminary</u> Let A, B, C be predicates — boolean expressions, if you prefer — in a number of free variables, then in

(0) $$[B \Rightarrow A]$$

$\Rightarrow$ denotes the logical implication and the square brackets denote universal quantification over all the free variables. (0) states that for every instantiation of the variables for which B is <u>true</u>, A is <u>true</u> as well. For example with $A \equiv x>0$ and $B \equiv x=1 \vee x=3$, (0) is <u>true</u>. Do we replace A by $x<0$, then (0) would become

<u>false</u>.

To stress that (0) expresses a reflexive binary relation between A and B, which is by the way a lattice order, we use from now on $\sqsubseteq$ (pronounced "under" or "weaker than") and rewrite (0) as

(1) $A \sqsubseteq B$.

The next step is to identify programs with the boolean expressions that characterizes their possible behaviours. If the program operates on N variables, the behaviour is expressed in terms of 2N variables, the dashed versions standing for the final values of the program variables.

Let, for instance, A be the (non-deterministic) program

<u>A</u>: x, y := x + any positive number, y

then its behaviour is fully characterized by

A: $x' > x \wedge y' = y$,

expressing that execution increases x by an unknown amount and leaves y unchanged. (The theory considers the 2 lines marked A: as alternative phrasings for the same predicate in the 4 variables x, x', y, y'. The one is just expressed in another language than the other.

Consider now a stronger program B, i.e. a B satisfying

(1) $$A \sqsubseteq B$$.

Then we call B "a refinement of A". An example for B would be (in program form)

$\underline{B}$: $x, y := x+1, y$,

and in boolean expression form

B: $x' = x+1 \wedge y' = y$.

Consider now the program C

$\underline{C}$: $x, y, z := x+1, y, z+y+x$

or, in boolean expression form

C: $x' = x+1 \wedge y' = y \wedge z' = z+y+x$.

Program C might strike you as an "extension" of B because it does something more: it increases z by $x+y$. But formally we have

(2) $$B \sqsubseteq C$$,

so C is a refinement of B. It would not have been so if we had not regarded program $\underline{B}$ as totally nondeterministic with respect to the unmentioned variable z.

Now this is unrealistic: the assignment statement is assumed to leave the value of all other variables unchanged and we would not accept a mechanism with uncontroled side-effects on z as a proper implementation of $\underline{B}$ (I think). Hoare & He have changed the boolean expression corresponding to the assignment statement by adding the conjunct

"and for all other variables α: $\alpha' = \alpha$"

(They express this differently.)

The proper way out is probably to prepare for the extension by already introducing the nondeterminacy that, if so desired, can be reduced by a refinement. Instead of $\underline{B}$ one should use

$\underline{D}$: $x, y, z := x+1, y,$ anything

to start with, thus a priori freeing z from the obligation to remain unchanged.

I hope that this conclusion doesn't bother you. It certainly does not bother me; I have always felt — since, say, EWD 227, that is — that programs had better be conceived as members of a family of related programs, and that the implied clairvoyance of the competent software engineer would cope with that.

Thank you for asking your question: it forced me to study the literature (which I enjoyed) and to rethink the assignment (what I enjoyed as well).

Greetings and best wishes,

yours ever,

Edsger