Copyright Notice

A somewhat open letter to EAA or: why I proved the boundedness of the non-determinacy in the way I did.

Dear EAA:

In your recent letter you wrote me about your doubts concerning the way in which I had proved that non-determinacy was bounded; you even feared that my arguments might be circular. Allow me to answer you in this somewhat public manner; I prefer to answer you in this way because you are not the only one who wondered why I proved it in the way I did.

I draw your attention to a paragraph from the last chapter of my book (p.213):

> "The next separations of concerns are carried through in the book itself: it is the separation between the mathematical concerns about correctness and the engineering concerns about execution. And we have carried this separation through to the extent that we have given an axiomatic definition of the semantics of our programming language which allows us, if we so desire, to ignore the possibility of execution. This is done in the book itself for the simple reason that, historically speaking, this separation has not been suggested by our rule of thumb; the operational approach, characterized by "The semantics itself is given by an interpreter that describes how the state vector changes as the computation progresses." (John McCarthy, 1965) was the predominant one during most of the sixties, from which R.W.Floyd (1967) and C.A.R.Hoare (1969) were among the first to depart."

Having quoted this paragraph I now feel tempted to add, that, at least in my own head, this separation of concerns took fully place while I was writing the book. In the fourth chapter the if...fi and the do...od are introduced by first giving an informal operational definition. What probably I should have stated more emphatically is that these operational descriptions should not be regarded as definitions upon which my definitions of the wp are based, but that these operational descriptions have been no more than a source of inspiration which can be forgotten as soon as the semantics for IF and DO in terms of the predicate transformer have been chosen.

If I choose to define the semantics of

do B1 → S1 [] ... [] Bn → Sn od

by $wp(DO, R) = (\underline{E}\ k:\ k \geq 0:\ H_k(R))$ with

$H_0(R) =$ R and non BB and $H_{k+1}(R) = wp(IF, H_k(R))$ or $H_0(R)$

then the weakest pre-condition $wp(DO, R)$ is given in terms of a recurrently defined sequence of conditions, and as such it has nothing to do with the notion of "repetition", a notion which refers to what might happen during execution by some implementation.

A usual argument to demonstrate the boundedness of the non-determinacy is via the class of possible computational histories. The argument is as follows. For a terminating computation each repetition is only executed a bounded number of times, in each alternative construct the computation is only of bounded non-determinacy, and, hence, by König's Lemma, the "computational tree" can only have a finite number of leaves (i.e. final states).

I have rejected the above argument for two reasons. First of all, it is based upon the consideration of the computational histories, whereas I want to ignore that my program texts also admit the interpretation of executable code. I have wanted to postulate the semantics independent of any underlying model of computation; I have done so, but then it is inelegant to prove such a fundamental property via such a model. I at least think it much more consistent to prove such a property directly.

The second reason, however, is that I think that the argument --at least as it stands-- is somewhat shaky. The problem lies with the justification of the suggested underlying computational model. After the postulation of the semantics, it is not too difficult (I think) to argue that the obvious implementation, when started in a state satisfying $wp(DO, R)$ will lead in a finite number of steps to a final state satisfying R . It is also clear (from the rejected argument) that then the number of possible final states is finite. But that could be a property of the implementation, viz. that it can only realize a finite number of the infinitely many permissible final states!

The only decent way I could think of is the one I have followed. First I postulate the way in which predicate transformers may be built up; next I prove the continuity of wp (via induction over the syntax); next I prove the boundedness of the non-determinacy (by deriving a contradiction from the

assumption of unbounded non-determinacy) and finally interpret this as a reason for reassurance (p.77):

> "A mechanism of unbounded nondeterminacy yet guaranteed to terminate would be able to make within a finite time a choice out of infinitely many possibilities: if such a mechanism could be formulated in our programming language, that very fact would present an insurmountable barrier to the possibility of the implementation of that programming language."

In other words: instead of "deriving" the boundedness of the non-determinacy from the possible behaviour of an implementation --whose "adequacy" must then be demonstrated in a rather complete way-- I prove the boundedness of the non-determinacy and remark that by doing so an otherwise unsurmountable barrier to the possibility of implementation has been removed. Note that nowhere in my book I have <u>proved</u> that my little programming language can, indeed, be implemented! That implementability seemed sufficiently obvious to me not to worry about it. Where is the suspected "circularity"?

> You write:
> "It concerns the definition of the semantics of the <u>do</u> construct (page 35). It seems to me that the semantics itself says that non-determinacy is bounded. It says that if a state satisfies wp(DO, R) , i.e. is bound to yield terminating computations finally satisfying R , then there exists a bound on the number of iterations of the DO for this initial state. This is reasonable since non-determinacy <u>is</u> bounded, but your <u>proof</u> of the boundedness uses the semantics of DO ."

Is it possible that you have suspected circularity by thinking that I have first taken implementability for granted, and then have made essential use of the implementability? Of course my proof of the boundedness of the non-determinacy uses the semantics of DO ! If I did not use the definition of the semantics, how could I prove something about them?

To think about the semantics of a programming language independent of of any underlying computational model is with our past, I admit, a difficult mental exercise. Perhaps you don't think it worthwhile. I personally think it is. As long as the operational approach remains the predominant one, languages for "sequential programming" and "concurrent programming" will remain two different topics. I hope to see these two topics merge into a

single one. I am hoping for a single programming language that allows sequential implementation, but also allows implementations displaying a lot of concurrency and allows that as "obviously" as the little programming language used in my book allows sequential implementation.

Logic has changed from a descriptive science into a prescriptive one; the "new logician" is an engineer. It is no longer the purpose of our programs to instruct our machines, it is the purpose of our machines to execute our programs. The semantics no longer need to capture the properties of mechanisms given in some other way, the postulated semantics are to be regarded as the specifications that a proper implementation should meet. As you may have concluded from the above, I have never been a great lover of automata theory!

Have I made myself clear now? I hope. The possible complaint against my book that the initial chapters of it don't make my position clear enough is a valid one: my attitude towards it subject matter evolved as a direct result of the very act of writing it! Perhaps --like many articles and most programs!-- also my book should be read backwards.

I thank you for your letter. Greetings and best wishes,

yours ever,

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow

Plataanstraat 5
5671 AL NUENEN
The Netherlands