Burroughs

PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

DR. EDSGER W. DIJKSTRA

Dr.H.Bekic
IBM Laboratorium Wien
Parkring 10
1010 WIEN
Oostenrijk

8th October 1974

Dear Dr.Bekic,

I received your letter of 2nd October 1974 yesterday and it distressed me to see that you found much of it quite upsetting. Perhaps one should not let other people peep in one's diary. I am not going to repeat here what I wrote to you in my personal covering letter: it still holds and I trust that you will honour my request.

In your letter you raise a number of questions, thereby challenging some of the opinions I expressed. The best I can do is to try to describe to you how I came to these opinions.

To start with: my suspicions with regard to defining the semantics of programming languages by means of an interpreter are of quite long standing, at least compared to the youth of our field. My first suspicion was roused by the observation that whenever we evoke via our program an available primitive --such as an ADD-instruction, say-- its occurrence in that program can only be justified by what that instruction achieves, by "what it does for you", rather than by "how it works". On the level of machine-code programming I could, for instance, not care less whether my machine has a serial adder or a parallel adder, as long as it adds. As soon as you decide to define what you mean by the sum of two numbers as the output of a carefully described adder when you feed it the operands as its input, you pay immediately a rather heavy price in at least two respects.

Firstly, the properties of addition --such as being commutative and

PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

DR. EDSGER W. DIJKSTRA

associative-- are fairly well hidden and you have to prove them as a property of your adder. I considered that a heavy price because, as far as the use of the ADD-instruction is concerned, it are exactly these properties that matter.

Secondly, it is practically impossible to give such a mechanistic definition without being over-specific. The first time that I can remember having voiced these doubts in public was at the W.G.2.1 meeting in 1965 in Princeton, where van Wijngaarden was at that time advocating to define the sum of two numbers as the result of manipulating two strings of decimal (!) digits. (I remember asking him whether he also cared to define the result of adding INSULT to INJURY ; that is, why I remember the whole episode.) And being over-specific seemed harmful to me in more than one respect. Firstly, one must state very carefully which aspects of the mechanism's behaviour are to be considered "essential" and which may be considered as "irrelevant", because without having done so it is impossible to consider an alternative, but in essentials equivalent alternative. Secondly, the presentation of an alternative implementation presents the problem of demonstrating the equivalence of algorithms, a problem which I did not regard as an attractive one.

At that moment, I could not offer van Wijngaarden an alternative technique for the definition of the semantics of programming languages, but feeling that the mechanistic definition did not seem to fulfill its most important functions too well, I started to think about why and how I would like to use the semantic definition. And as a result I started to think about how to prove the correctness of programs --not all the time! I had other axes to grind as well!-- . And I discovered the following. At that time considerable attention had been paid to the question how to decide, whether two given programs were equivalent or not. (For this fact I can think of two explanations. It was one of the ways in which the problem of compiler correctness came to us; besides that, the boolean question "equivalent or not" has a certain mathematical appeal.) Very little attention --as far as I could find-- had, however, been given to the question how to design programs in such a systematic

PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

DR. EDSGER W. DIJKSTRA

way that they would be correct by virtue of the way in which they were constructed. That second question, however, seemed to me more relevant --after all: programs are not "given",they have to be "constructed"-- and also more tractable. As the result of some exercises I wrote an article "A constructive approach to the problem of program correctness." (It was published in BIT: if I remember the dates correctly, it was written in 1967 and published in 1968.) Much of the argument was still fairly informal, all sorts of hand-waving included, but at that time I did not worry about that: before developing a formalism I wanted to get a feeling of the demands that would be made upon that formalism. About that time, three articles appeared that gave a first start towards such a formalism: Peter Naur's article about the General Snapshots, Robert Floyd's article about Assigning meanings to programs, and, a year later Tony's article on the axiomatic definition in the Comm.A.C.M.

I think that at that time, Peter's article appealed most to me. From my reaction to Bob's article I remember that I was rather put of by his use of flowcharts and the resulting need to isolate cycles in the corresponding graph. Tony's article attracted me in the sense that it tied in with the syntactical structure of the program text, but the separation in axioms and rules of inference --a logical tradition, but not my tradition-- worried me. My first reaction to these three articles was "Interesting, wait and see...".

In the next years, Scott's Mathematical Semantics began to appear. I found that very difficult when I first saw that on account of sheer lack of mathematical knowledge on my side. I studied a certain amount of it and also decided "Interesting, wait and see...". For a considerable period of time I hesitated whether I should develop my agility in the propositional calculus, or get a thorough knowledge of the ins and outs of lattice theory or both. I decided to wait another half year before tentatively making up my mind, and to go my own way in the mean time.

As a side-effect of lectures I gave during that period of time, I

PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

DR. EDSGER W. DIJKSTRA

learned how to develop in a systematic way the synchronization conditions needed to ensure a given form of harmonious cooperation between loosely coupled sequential processes. I became more and more constructive in my approach and the experience at that time is, I think, responsible for the fact that what for a number of years had been a matter of personal policy, now became a firm principle, viz. never to maneuver myself in the position where I would have to derive the unknown properties of a given algorithm, nor to try to prove the correctness of a given algorithm a posteriori. Both problems struck me as putting the cart before the horse. When, later, I saw Wegbreit's article in the Comm.A.C.M, in which, for instance, he tried to find the invariant of a loop, I felt fully justified in my decision to avoid that problem.

In the mean time the balance slowly started to tip over in favour of the axiomatic method compared to the mathematical semantics. While Tony's original article presented axioms or postulates about programming constructs of which it was not fully clear --at least to me-- to what extent they could take the place of a definition of the semantics and while they only enabled the deriviation of sufficient pre-conditions for partial correctness, that has been cleaned up and one can now give --very much in the same style-- the weakest pre-condition for total correctness and as a system for semantic definition it now seems fully satisfactory. (In particular, all arguments in favour of "complementary definitions" as suggested by Hoare and Lauer in Acta Informatica seem to have disappeared --this is not only my impression: Tony confirmed it when I spoke to him last September--; thank goodness, for I never liked to complementary definitions: that was another article of which I said to myself "Wait and see...." but this time in the hope that it would become superfluous.) Another reason, why the balance started to tip over was that the probability that the theory could be forged into a tool, helpful for actual program construction, seemed greater for the axiomatic method than for the mathematical semantics.

PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

DR. EDSGER W. DIJKSTRA

The next tipping of the balance took place during the meeting of W.G.2.3 in Munich in April 1973, where John Reynolds gave for our instruction a commented survey of the mathematical semantics. It then came towards me as a view of computing that can be appreciated for its consistency, for its conceptual unification, but not without a few drawbacks, which I am beginning to regard as serious. The first one is the size of the mathematical machinery involved, compared to which the propositional calculus --and for an axiomatic definition now nothing more seems needed-- is negligible. Of course one can point out that up till now the axiomatic method is quite modest in its aims --e.g. no functions as values--, on the other hand, within that limited scope (and I am greatly in favour of choosing our limitations wisely) the cost/performance ratio --if you allow me the use of that term as one computing industrialist writing to another one-- of the two different approaches seems to be very different. The second drawback is that defining the semantics as a minimal fixpoint is a very indirect one --some people would even call it clumsy--: first one needs functional analysis in order to introduce the concept of a fixpoint, and after that lattice theory to formulate that it is not just any fixpoint, but among the fixpoints the minimal one.

A further reason that tipped the balance was our realization at that Munich meeting, that one of the usual motivations for the mathematical semantics does not hold water, viz. that the study of infinite computations is of importance for "operating systems and airline reservation systems". (We heard it repeated in Newcastle, if I remember correctly by Milner.) John Reynolds started in Munich also with the operating systems and the airline reservation systems and after that two-minute motivation.... he talked for the remainder of more than an hour about the logical difficulties of dealing with real numbers! It was a very nice and illuminating talk (at which I rejoiced that real numbers are no longer my problem!), but by the time it is suggested that one needs real number theory for dealing with operating systems and airline reservation systems, some hilarious mistake has been made. (Actually, that a theory for dealing with operating systems and airline systems can

PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

DR. EDSGER W. DIJKSTRA

shed some new light on the real numbers is perhaps more probable.) The inclusion of infinite computations should have a better motivation, for instance for the sake of simplification of one's arguments (the invention of projective geometry by extending the Euclidean plane with its horizon is an example, where such an extension made a drastic simplification possible). Such a simplifying breakthrough, however, was not transmitted to me. It may happen one day --I try to keep my mind open-- but I expect it less and less the more I realize to what extent the simplicity of the axiomatic method is dependent on the explicit restriction to computations which are guaranteed to be finite.

Other facts that I take into consideration when trying to estimate significance and potential of a scientific theory are the problems it evokes and the problems it is applied to. You write to me in your letter "Nondeterminism is a phenomenon we are all interested in and the question how to deal with it in any given semantical framework seems to be a legitimate and interesting one." Of course, as long as you allow me to draw at least some conclusions about the apparent adequacy of such a "given semantical framework" when the dealing with nondeterminism presents any more or less serious problems: the drawing of such conclusions seems to me the major purpose of such an exercise.I could not help comparing your struggle with the current state of affairs in the axiomatic method, in which non-determinism --as a matter of fact: to the extent of the non-determinism as displayed by a general, finite Petri net-- enters the picture so naturally, that I have come to regard determinism as a special case (which, by the way, does not seem to be very interesting either, because, as far as mathematical manageability is concerned, there is hardly anything to be gained by restricting oneself to the fully deterministic case). I could not help to compare, could I?

Furthermore, during the last decade there seems a shift of emphasis to have taken place, a shift of emphasis that seems worthwhile to draw your attention to, as also such an emphasis plays a role when trying to estimate

PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

DR. EDSGER W. DIJKSTRA

significance and potential of a scientific theory. While in the beginning of formal definitions of programming languages the efforts were very much directed upon the needs and problems of the language implementer, this emphasis shifted during the next ten years via the problems of the language designer towards the needs of the language user. Taking into account that the combined users present the broadest interface, I think we must regard this as a healthy and welcome development. But such a shift has a profound influence on our appreciations.

For instance, in the most recent Computing Reviews I found (CR27.102) Zemanek being quoted, when he observed that an axiomatic method only works for a "highly elegant environment". I hope that he is right. Under the assumption that he is, ten years ago this observation would have been interpreted as one of the shortcomings of axiomatic methods. Now, however, I think that we react inversely and say "One up for axiomatic methods!": at last a discipline which, when you stick to it, forces elegance upon you! Great! And if I am then exposed to a lecture in which Scott has chosen to demonstrate the descriptive power of his method quite emphatically with an example including goto-statements, what do you think that happens to the balance? It, again, tips a little bit further. There is between the two of us perhaps a profound difference in appreciation of the "power" of a descriptive method: I do not care for the ability of describing things, of which I know already on other grounds that they had better be left alone, stronger: I rather had not such an ability.

You see, that I am answering your letter backwards. I have not "dismissed the whole thing" (yet), but it is undeniable that the balance made a further tip. And I trust that you now understand why.

* * *

On page 1 I described to you my first reasons for being suspicious as regards mechanistic definitions of semantics. As time went on, that

PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

DR. EDSGER W. DIJKSTRA

suspicion grew. Defining the semantics of a programming language by means of an interpreter for it, defines the meaning of algorithms written in that programming language by means of ... another algorithm! As such, it does not solve the problem, it only pushes it one stage further. But even if we assume to be able to understand --or "agree upon"-- how the interpreter is supposed to act, what then can we do with that interpreter, when faced with a program written in the language concerned? Well, the interpreter can interpret the program, provided we choose values for the input. But that gives us no more information than program testing, and such sampling will never enable us to make assertions pertaining to "any" input value of the domain. So how can we proceed? Well, we could try to prove --how, is somewhat mysterious, but for the sake of the argument we can suppose that we can-- certain properties of the interpreter on account of which we can prove properties of programs being interpreted, which, of course, must depend on the program text. But seems this a sensible approach? I do not think so.

Firstly, the mystery of how we can prove such properties of the interpreter bothers me, but secondly, it seems a very tortous way of deriving properties of the programs interpreted, because if those are the properties we are after, why not postulate directly, how they depend on the program text and forget about the whole interpreter? That seems much more efficient. Instead of interpreting a program text as a piece of executable code, we can also give it a completely timeless interpretation and introduce axiomatically how each program text defines a relation between the initial state space and the final state space. (If I am in a very puritan mood, I even talk about the left-hand state space and the right-hand state space, just in order to remove all connotations with sequential execution.)

If you start to think about it, the advantages of the latter approach are overwhelming. It gives, for instance, a clear distinction between the semantics of the programming language proper --i.e. how the text defines

PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

DR. EDSGER W. DIJKSTRA

a relation between the left-hand state space and the right-hand state space--and its implementation. Correctness concerns belong to the realm of the semantics of the programming language proper, efficiency concerns have only a meaning with respect to a specific implementation, and nothing is gained by mixing these concerns, on the contrary.

In one mood I can regard the program text as a mathematical object and convince myself that, according to the rules of the semantics it defines the desired relation between left-hand state space and right-hand state space: time, sequencing, alternative actions, repetition, determinism, intermediate machine states and all that, none of it needs to enter my mind, when I am in that mood (if it does, it is only confusing, because irrelevant as far as correctness is concerned). In another mood, I can ponder about the price of execution by a given implementation, for instance by knowing that, what in my first mood was something like functional composition, indicated in the text by, say, a semicolon, corresponds in that implementation to a concatenation in time of two activities, separated by "an intermediate state of that machine". In that second mood, however, I need not worry about the question whether the execution will produce the desired result.

The grave disadvantage of defining programming language semantics by means of an interpreter is that it defines the relation between left-hand state space and right-hand state space as the result of a computational process, as the last of a long series of machine states, while during correctness considerations, of this long series only the first and the last matter. But if the intermediate ones don't matter, please define your semantics in such a way that they don't enter the picture to start with. As far as thinking about programs and programming languages is concerned, that is so much more efficient. Distinguishing in one's mind between a programming language and its implmentation(s) is already difficult enough, (as is illustrated by not uncommon assertions such as "ALGOL 60 is an in-

PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

DR. EDSGER W. DIJKSTRA

efficient language" or "PL/I pointers are more efficient than ALGOL 68 references sentences so elliptic as being close to nonsense).

When I first heard that the Vienna crew was tackling a formal definition of PL/I, I was horrified, because on account of what I knew of it, PL/I seemed to me one of the most unattractive objects to give a formal definition of. I certainly did not envy them their job. The next thing I heard was that a definition by means of an abstract interpreter was chosen, a message to which I could give only one interpretation, viz. that, indeed, PL/I was a most unattractive object to give a formal definition of. Without further knowledge I have assumed without hesitation that "an interpreter" was at least your second choice, as I could not envisage the Vienna crew not sharing a good fraction of my misgivings about mechanistic definition of semantics. (And I still believe that they are shared by many of its members.) And when I received what is now known as "The Vienna Telephone Directory", I was once more horrified; also very much impressed. I had not the slightest intention of being "unfair" to the Vienna crew and, if you smelled criticism, it was criticism regarding PL/I rather than VDL. But you understand that the whole project struck me as relatively successful attempt to make the best of a bad job. That "at least with metalanguages we should be given a chance to correct our mistakes" is a prayer that has my full sympathy, I would even like to extend the prayer to programming languages.

With some of the points raised in the preceeding paragraph I have more or less dealt, but not with your suggestion that eventually the study of such a merging operator is unavoidable. Is it? And, if so, has it a place in the definition of the semantics of programming languages? I am not so sure. In the case of what is usually called a sequential programming language, we have seen that as far as its semantics are concerned a completely time-less interpretation suffices, so time-less as a matter of fact that even the notion "before-after" has no place. For the sake of convenience one is usually not a Puritan and one talks about initial and final state, about pre- and post-

DR. EDSGER W. DIJKSTRA PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

conditions, but I found it most convenient to start at the post-condition. (To follow the relation the other way round is something I gladly leave to the machines!) Stronger: on that level it is totally irrelevant that one's primitives and the way in which they may be combined, are such that implementation by a sequential machine is absolutely straightforward. The usual consequence of such choices is that implementation by a machine that we would like to display more concurrent activity is less obvious.

One can certainly think about a programming language for which the primitives and the way in which they may be combined have carefully been chosen in such a way that implementation by a "less sequential machine" becomes equally obvious, but again: the way in which its semantics have been defined need (and I think, therefore: should) in no way reflect this fact. It is not unthinkable that the implementor can exploit nondeterminism of the programming language by omitting some synchronization constraints, leaving speed ratio's rather undefined, you name it. But starting at this end the non-determinism has already been introduced in the usual noiseless fashion and one is never faced with the question what all possible sequences may do in the sense that, whatever sequence will be chosen, one has already established that what will happen will be acceptable, so one does not bother about what happened. The merging operator only needs to be studied if one intends to use it to generate non-determinism that one can then study. I should rather start with the non-determinacy at the semantic level and later exploit it while implementing.

Compared with the mechanistic approach this is, of course, in the other way. I think it is easier. The only thing the traditional logicians did was to try to find a model for the real world, but, since in the form of computing science logic has also become an engineering activity, I prefer the real world to provide a model for my dreams....

As to the preceeding paragraph of your letter: please don't be worried

DR. EDSGER W. DIJKSTRA PLATAANSTRAAT 5 NUENEN THE NETHERLANDS

by the fact that in your first lecture you more or less failed to reach your public: not all our kites fly.... (As you have seen, more or less the same happened to me in Edinburgh the day before. Of course one regrets it, but one should take the risk: it is vain to pretend all one's talks are a guaranteed success.)

In view of the fact that you have sent a copy of your letter to nine gentlemen, I guess, that I should do the same with my answer to you. I hope that my explanations in this letter have made the trip report less upsetting. Please don't feel guilty on account of the length of this answer. It was a pleasure to write to you and to show to you, by doing so, my appreciation for the fact that you wrote to me in the first place.

With my best wishes and warmest regards,

yours ever

Edsger

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow

cc.: R.M.Burstall
C.A.R.Hoare
J.Hopcroft
E.S.Page
M.Rabin
B.Randell
D.S.Scott
S.Winograd
H.Zemanek