On language constraints enforceable by translators.

(An open letter to Lt.Col.William A.Whitaker.)

Friday 3rd March 1978

Dear Colonel,

this letter is an almost immediate reaction to the 4 kilogram of language design documentation that reached me last Monday; it is written because --at least in those few days-- I failed to discover an adequate treatment of an issue that now seems to me to be in urgent need of clarification. My comments are pertinent to the last sentence of Requirement 1F (Revised "Ironman", July 1977):

"There shall be no language restrictions that are not enforceable by translators."

and I shall illustrate them by a short discussion of side effects.

Ironman's requirements 4C and 7E don't encourage side effects and the inclusion of these requirements reflects the general recognition of the undesirability of side effects. Despite this general recognition of their undesirability Ironman, however, didn't dare to rule them out! The only explanation I can think of is the fear --or knowledge-- that, on order to be enforceable by translators, language restrictions preventing side effects would be too severe to live with, and would throw away the child with the bathwater.

* * *

Let us first try to capture formally what the statement --whether theorem or postulate-- "the integer procedure f is free of side effects" should mean. (For the sake of simplicity I use in this discussion here an integer procedure without formal parameters that, when called, delivers in general a value that is functionally dependent on the initial values of some of its global variables.) My first proposal is "the integer procedure f is free of side effects if and only if (within its scope) the inner block --ALGOL 60 conventions--

<u>begin</u> <u>integer</u> h; h:= f <u>end</u>   (1)

is semantically equivalent to the empty statement".

This choice is justified by the circumstance that when f is free of side effects according to this definition, the following are transformations an optimizing compiler might undertake as harmless:

1) Transform y:= f * f into _begin_ _integer_ h; h:= f; y:= h * h _end_
(the above as paradigm for "taking a constant expression outside a loop")

2) Transform b _or_ f = 1 into _if_ b _then_ _true_ _else_ f = 1

3) Transform a * f into _if_ a = 0 _then_ 0 _else_ a * f

Without severe --and we may safely state: unrealistically severe-- restrictions on the text of the procedure f , it is _impossible_ for a translator to "enforce" that the function procedure f is in the above sense free of side effects, as it would require the solution of the halting problem, a problem of which we know since 1936 thanks to A.M.Turing that in general it is unsolvable. Statement (1) is _not_ equivalent to the empty statement under all circumstances in which the calling of f leads not to a properly terminating computation!

In general f computes a partial function and calling f only leads to a properly terminating computation provided some condition D --describing its domain-- is initially satisfied. Thus we come to the definition:

"The integer procedure f is free of side effects with respect to condition D if and only if

"_if_ D _then_ _begin_ _integer_ h; h:= f _end_ _else_ skip"

is semantically equivalent to the empty statement."

Note. The fact that the condition is not necessarily identically true need not be caused by the possibility of nontermination. Derive --under the assumption of a variable x global to f -- from the text of f the text f' by inserting the statement "x:= abs(x)". Then D' = D _and_ $x \geq 0$. (End of note.)

In general the condition D with respect to which a function procedure is free of side effects needs to be stated explicitly; and the user of the function procedure has to ensure that this condition is satisfied wherever the function procedure may be invoked.

* * *

The contrast between the following two function procedures with "own variables" may shed a further light. I guess we all abhor a random function like --described in ALGOL 60, extended with initialization of own variables--

```
real procedure random;
  begin own real h = some initial value;
        h:= some scrambling of(h);
        random:= h
  end
```

It could be prevented by ruling out function procedures with own variables of any sort, but that could be regarded as throwing the child away with the bathwater, as it would also exclude the so-called "memo-function" mf --ALGOL 60 now extended with a while - do construct--

```
real procedure mf(x); real x; value x;
  begin own real X = some initial value;
        own real F = f(X);
        while x ≠ X do X:= x; F:= f(X) od;
        mf:= F
  end
```

for which $mf(x) = f(x)$ holds thanks to the internal invariance of $F = f(X)$

Assuming that something like "random" should be out and something like "mf" should be in, we can only conclude that we need to distinguish between the notion of "a <u>legal</u> program" and the notion of "a <u>correct</u> program". From translators we can require that they reject illegal programs; <u>for legal programs the language definition should define the proof obligations to be met in order to make the legal program also a correct program</u>.

A checking compiler may assist the programmer in fulfilling these proof obligations by detecting cases were the proof is clearly impossible --as in the case of "random"-- or detecting cases where the proof is trivial. (The latter seems to me to be the main function of scope rules.)

* * *

As far as I have been able to detect in those few days, the four language designs have only paid lip service to the formal aspects of programming. My comments in margine next to requirement 1H, first sentence:

> "To the extent that a formal definition assists in achieving the above goals, the language shall be formally defined."

was "a compromise"; from competent language designers we could, however, have expected the awareness that a programming language definition should define an interface between the mechanics and their users, i.e. should give a clear definition of mutual rights and obligations.

To require from the mechanics the rejection of all incorrect programs is as naive as requiring the Philosopher's Stone: it would impose upon implementors the obligation to include all the theorem proving techniques yet to be developed by the Artificial Intelligentsia plus the nonexistent solution of the halting problem. Hence the <u>necessity</u> to distinguish between the notions "legal" and "correct", and hence the necessity to define what else is required that a legal program be also a correct program. The adoption of any language the definition of which fails to do so will only perpetuate the muddle and confusion (with all the unavoidable expensive consequences) the DoD seeks to overcome.

If I were to select which of the four tenders should be granted a phase 2 contract, I would try to estimate their respective abilities to take the above formal requirement seriously and to act accordingly. Rewriting the requirements and starting all over again could be more fair and sensible: requirement 1H is such "a compromise" that it almost excludes a language of the quality the the DoD needs. (This is the first conclusion I was led to by the study of the four proposals.) I wish you the courage of reconsideration and the political tools needed for the prevention of another billion-dollar mistake.

Yours sincerely

Edsger W. Dijkstra

prof.dr.Edsger W.Dijkstra DSc
Burroughs Research Fellow

Plataanstraat 5
5671 AL NUENEN
The Netherlands