On the GREEN Language submitted to the DoD.

"Parts of it are excellent!"
(Re "The Curate's Egg", PUNCH, vol.cix, p.222, 1895)

I found the GREEN Language very hard to penetrate. The main source of my difficulties has probably been that I tried to reconstruct from the documentation a coherent design philosophy, an effort in which I failed: the mixture between sense and nonsense remained baffling. Eventually I had to assume that upon a defensible core, designed by one group, presumably another group had grafted all sorts of goodies inspired by specific Ironman requirements (or some similar process).

It gives one hope to read in the Rationale (RAT i) "The reference manual contains a complete and concise definition of the language. Following Wirth we believe in the virtue of having a rather short reference manual." I follow in this respect Wirth too: my main complaint about ALGOL 60 is that its syntax is too baroque and the Report on the Algorithmic Language ALGOL 60 is too long. Reading the Reference Manual (RM) for the GREEN Language, however, one discovers to one's disappointment that it is neither complete, nor concise.

Its Introduction (RM p.1) expresses "a deep concern for programming as a human activity" and "an attempt was made to keep the language as small as possible." But was it kept small? We find (RAT p.1-2) "Reserved words are distinguished from program identifiers. There is a small list of 72 (sic!) such words." That is a vocabulary of more than 10 percent of Basic English!

I thought that it was a firm principle of language design --out of concern for programming as a human activity-- that in all respects equivalent programs should have few possibilities for different representations (possibility for differences ideally not going beyond the arbitrary choice of identifiers and the arbitrary ordering of semantically unordered components). Otherwise completely different styles of programming arise unnecessarily, thereby hampering maintainability, readebility and what have you. This requires from the language designers the courage to make up their minds! The designers of the GREEN Language have repeatedly lacked that courage, and have provided multiple ways of doing the same thing.

Unable to choose between the specification of actual parameters by position or by naming them via the identifier of the formal parameter, they provide both options! Unable to choose between "while more_to_do loop" and "until goal_reached loop" they provide both. They do so with the extremely weak justification (RAT 1-24) "Its purpose is to avoid the inconvenience and obscurity (sic) of negations." From the point of view of homogeneity this proposed justification is not just "extremely weak", but invalid.

Similarly they have refused to make up their mind with respect to the ambiguities that may arise when the same identifier happens to be used as component of different enumeration types. In RAT p.1-14 they suggest "In some contexts it may be necessary to qualify the type of GREEN by writing PAINT(GREEN) or TRAFFIC_LIGHT(GREEN).", in RM p.46 they suggest to resolve naming conflicts by renaming clauses.

* * *

Another disturbing feature of the Rationale is the superficial nature of many "justifications". To "justify" the absence of conditional expressions by noting that (RAT p.1-5) "conditional expressions [...] pose severe problems for automatic layout" is original but ridiculous. It is for fundamental reasons very regrettable that the Ironman requirements seem to require an explicit statement terminator rather than a separator. (Nobody seems to have remarked that all arguments for an explicit terminator can also be turned into arguments for an explicit initiator!) However, when they write "Extensive analysis of programmer errors supports the use of the semicolon as a terminator [GH 75]." (RAT p.1-5) they damage the standing of their proposal, for the experiments described in the paper quoted are too lousy to justify any conclusion at all. When they write (RAT p.1-25) "It was also felt that whereas nondeterminism is in the essence of parallelism, its introduction for sequential programming would not appear natural to most users." (my underlining) they reflect several misunderstandings in a single sentence. First of all they ignore that it has been recognized already several years ago that nondeterminism and concurrency are separate issues in the sense that both nondeterministic programming languages whose obvious implementation does not introduce concurrency and deterministic programming languages whose implementation obviously admits concurrency are not only quite conceivable, but even worth of our attention. Secondly they confuse in their appeal to what seems "natural to most users" the notion "con-

venient" with the notion "conventional".

* * *

After having referred to the difficulties of the types in PASCAL, they write (RAT p.1-8) "These problems are overcome in the GREEN Language with the notion of subtype." Are they? They first explain the introduction of new types with "type weight = integer; type length = integer" which "define the new types "weight" and "length" as different types, both distinct from the predefined type "integer" although they have the properties of "integer"." That is a very obscure sentence: if the new types have all the properties of the old type, how can they differ? They give two examples of assignment statements that are supposed to explain this by their being invalid; however "two "weights" or two "lengths" can be added in the normal way". What is the type of such sums? Presumably again "weight" or "length" respectively. Are we allowed to multiply a "weight" with a "length", and is the result a "torque"? I find that section of the Rationale obscure and misleading if it suggests that a program can be enhanced for greater security with some sort of "dimension analysis" that will be checked by the compiler. This problem is much more difficult than suggested by the remark that you may not assign a "weight" to a "length". (A recent error analysis of student programs revealed an unawareness of the difference between "a moment" and "a period (of time)" —a confusion induced by the Dutch language, in which the most common term can be used for both— ; as a result some students had failed to realize that the sum of two periods again gives a period, but the sum of a moment and a period a new moment. It was my original training as a physicist that enabled me to grade these programs in a single glance as "wrong", but I wouldn't venture to build that in into a language and its compiler.) In short: the rest of RAT p.1-8 is hopelessly superficial; the pages RM p.23/24 were not of much help either, as the expressions "same type" and "identical type" are not defined. The language seems to be "strongly typed", but only to the superficial reader;as it stands this way of introducing new types does not seem to serve a single purpose. I at least couldn't discover one.

Also the notion of subtype present some problems to me. It is explained that the declaration "subtype SMALL_INT = INTEGER range (1..100)" makes "the declarations:

```
I: SMALL_INT;                    case A
J: SMALL_INT;
```

equivalent (my underlining) to the declarations

```
I: INTEGER range(1..100);        case B
J: INTEGER range(1..100);   "
```

Consider now in addition

```
"subtype FUN_INT = INTEGER range (1..100)
 K: FUN_INT;
 H: INTEGER range (1..100)"
```

Combining H with case B I am tempted to conclude that H, I, and J are all of the same type and subtype. On account of the equivalence we conclude that this is also the case when H is combined with case A ; hence, if we so desire, we can conclude that the declarations of case A and case B are also equivalent with

```
I: SMALL_INT;                    case C
J: INTEGER range (1..100)
```

In the same vein we conclude that the type and subtype of the variables H and K are the same. On account of the transitivity of the notion "the same" I draw the same conclusion for K and I and J from case A . But how do we reconcile all this with (RM p.18):

"Declarations of distinct type names always denote distinct base types, even if their definitions are identical." ?

The way out of this muddle seems to regard my underlined "equivalent" as a mistake; that would invalidate the above argument. But that does not help, for the sentence just quoted is also hard to reconcile with what the Reference Manual says two sentences further

"The base type of a subtype is that of its parent type."

In short: the claims that the Reference Manual gives a "complete and concise" definition of the GREEN Language, and that PASCAL's type problems "are overcome in the GREEN Language" seem to me idle boasts.

* * *

One of its nicer aspects is presented by the synchronization primitives, their "boxes". When I studied the Reference Manual --RM p.52-- I became very suspicious because a standard bounded buffer seemed to require three boxes, one between producer and buffer, but two between buffer and consumer. Because --as is well known for about fifteen years-- as far as synchronization is concerned the relation between producer and consumer is totally symmetrical, I became very suspicious indeed. Upon closer scrutiny the asymmetry was not an unavoidable consequence of the synchronizing primitives used, it was just the result of lousy coding: it is easy to program a buffer that only needs two "boxes" --and it is more efficient too!-- .

The language contains an unjustified constraint, which, however, seems very wise. Local nondeterminacy in communicating programs may be resolved by the timing of occurrences in other paths: the nondeterministic program has its nondeterminacy concentrated in a select statement, that can only be controlled by "send" and "receive" , but not by "connect" . This may surprise readers, the constraint is fully justified: it rules out coincident nondeterminacies in more than one path, nondeterminacies that have to be resolved consistently.

So that is nice. It has a few less attractive features too. With each box a first-in-first-out queue is associated, and that is very fine when first-in-first-out is exactly what you need. The major problem for a monitor addressed via such boxes is that it has only access to the oldest request for each box: honouring the request implies a coincident waking up of the process at the other side, whether this is desirable or not. As soon as I tried to implement a very different scheduling strategy I found myself forced to introduce for the monitor a linear array of boxes of a length at least equal to the number of processes to be monitored.

From an esthetic point of view that is not attractive: many monitoring algorithms allow a formulation that is absolutely independent of the number of processes to be monitored. Now this number has to appear somewhere in the text of the monitor, we have a constraint that will be hard to enforce by the implementation. Even worse: if the number of processes increases beyond the maximum that was foreseen, the monitor needs to be recompiled!

Two remarks are to be made in this connection. The queues associated with each box of the linear array I had to introduce have at most the length 1 --as in Concurrent Pascal-- and one of the people who saw me programming that monitor immediately recognized the style he had been forced to adopt while using Concurrent Pascal and that he had learned to dislike very much. It seems that most of the ill effects of one of the major shortcomings of Concurrent Pascal have been reintroduced with the boxes of the GREEN Language.

The second remark concerns its implications for Generic Program Units. In the justification the authors write (RAT p.5-43) "Fortunately during the same period of time another important question in the language design was discussed in detail: the concept of generic. We had there a very good solution for our practical problem." and (RAT p.5-44) "Notice, and it is fundamental, that these instantiations of the generic RW are made at compile time, like all generic instantiations, and not at run time: we are still consistent with our first design decision." (i.e. the decision that the degree of parallelism is settled at compile time). Well, that first design decision is defensible, and I can understand the designers enthusiasm. They have failed to point out that the decision of the multiplicity is not necessarily only reflected in the number of declarations declaring an instance but may have other consequences, as soon as an upper bound for the multiplicity has to appear as compile time constant in other places, other definition modules even!

According to the buoyant language used in the documents it must be a pure joy to recompile separate program components written in the GREEN Language. From my side of the Atlantic Ocean I can only express the fear that special constraints incorporated in the GREEN Language will make such recompilations uncomfortably often necessary. The notion of compile time constants is of course the villain in the piece --see EWD659 for more details-- ; on account of the above observation I see very little justification for the hope that the obligation to recompile will not frequently propagate all through the system.

I trust others more capable than myself of pointing out eloquently that the pragmat will open Pandora's box, and that the ills escaping will not be cured by the comforting knowledge that a pragmat (RM p.6) is "terminated

by the end of the line". I must, however, make one exception. I quote (RAT p.5-45) "Simulation is entirely transparent in the language. When using the pragmat SIMULATION, the system (sic) runs under simulated time: the real time clock is implicitly replaced by the simulated time clock." How do the gentlemen propose to perform a simulation in combination with real-time obligations? They certainly know that such a combination is not unusual at all: it is the quintessence of computerized observation of external dynamic systems! I am absolutely at a loss if I try to understand how such a pragmat can be honestly proposed. I regard this proposal as a complete disqualification of the authors and, hence, of their proposal.

When I came down for dinner and my wife asked me how things were, I could only summarize: "Technical incompetence, probably enhanced by dishonesty."

Plataanstraat 5
5671 AL NUENEN
The Netherlands

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow