A review of the 1977 Turing Award Lecture by John Backus.

The last sentence of the first section (titled: "Conventional Programming Languages: Fat and Flabby") of this paper [1] captures it well:

> "The purpose of this article is twofold; first, to suggest that basic defects in the framework of conventional languages make their expressive weakness and their cancerous growth inevitable, and second, to suggest some alternate [meant is "alternative", EWD] avenues of exploration toward the design of new kinds of languages."

Of the 28 pages, about a quarter has been devoted to the first purpose, and for a justification of the research described in the rest of the paper that seems a bit too much of a good thing. (By way of comparison, in the Ph.D thesis of Martin Rem [2], who also explored a new kind of language, the following justification sufficed:

> "Present-day programming languages reflect present-day technology. New techniques --associative addressing, large scale integration (LSI)-- are being developed. These new techniques may very well allow for a truthful implementation of radically different programming languages.")

Besides being too long for a justification, that first part is also in other respects not fully satisfactory. It bemoans the clumsiness engendered by the conventional architecture ("In fact, conventional languages create unnecessary confusion in the way we think about programs." and "Surely there must be a less primitive way of making big changes in the store than by pushing vast numbers of words back and forth through the von Neumann bottleneck."); but in the second part of the paper, where Backus sketches an alternative, matters of implementation are hardly touched upon ("Efficiency questions are beyond the scope of this paper.") He presents "the present condition of obesity" of today's programming languages almost as a historical necessity --a kind of reasoning I have learned to mistrust since World War II-- with "the von Neumann bottleneck" as its necessary and sufficient cause. But isn't that a bit too much of a simplification? (He writes "The Department of Defense has current plans for a committee-designed language standard that could require a manual as long as 1,000 pages." as if that manual would then be the only over-elaborate document within the DoD!) He writes that "smaller, more

elegant languages such as Pascal continue to be popular", where in the case of Pascal "rapidly gaining in popularity" would be more accurate. He presents the proving of the correctness of programs as an activity reserved for geniuses: "The complexity of this axiomatic game of proving facts about von Neumann programs makes the successes of its practitioners all the more admirable. Their success rests on two factors in addition to their ingenuity". And then comes his fundamental complaint "In any case, proofs about programs use the language of logic, not the language of programs. Proofs talk about programs but cannot involve them directly [? EWD] since the axioms of von Neumann languages are so unusable." and he presents as an advantage --without questioning-- that in his system "Algebraic transformations and proofs use the language of the programs themselves, rather than the language of logic, which talks about programs." I am not quite sure what is meant by talking proofs and talking logic. But whereas machines must be able to execute programs (without understanding them), people must be able to understand them (without executing them). These two activities are so utterly disconnected --the one can take place without the other-- that I fail to see the claimed advantage of being so "monolingual". (It may appear perhaps as an advantage to someone who has not grasped yet the postulational method for defining programming language semantics and still tries to understand programs in terms of an underlying computational model. Backus's section "Classification of Models" could be a further indication that he still belongs to that category. If that indication is correct, his objection is less against von Neumann programs than against his own clumsy way of trying to understand them.)

The rest of the paper is devoted to "Functional Programming Systems" (9 columns), "The Algebra of Programs for FP Systems" (15 columns), "Formal Systems for Functional Programming (6 columns), "Applicative State Transition Systems" (8 columns) and 4 columns remarks and summary.

Functional Programming Systems are characterized by:

> "An FP system has a single operation, application. If f is a function and x is an object, then f:x is an application and denotes the object which is the result of applying f to x .
> f is the operator of the application and x is the operand."

Objects are "bottom" (or "undefined"), atoms, or sequences of objects, and a whole set of primitive functions is suggested that inspect, shorten, extend, merge, distribute, and massage objects. Most of these operations are defined in terms of rearranging and/or deleting and/or creating multiple copies of sequence elements. Collectively I shall call them "shunting operations".

Next a set of Functional Forms is given, i.e. ways of combining or modifying functions into new functions --Composition, Construction, Insert, Apply to All (which applies a function to all elements of its operand sequence separately) etc.-- . The examples are very traditional (factorial, inner-product, matrix multiplication); but already the matrix multiplication --a problem that seems specially designed for such systematic shunting operations-- displays what seems characteristic for this style of functional programming. In order to transform $\langle m, n \rangle$ into $\langle m, trans{:}n \rangle$ we must apply the function $[1, trans \circ 2]$ which leads to the following computational steps:

$[1, trans \circ 2]{:} \langle m, n \rangle =$ (Construction)
$\langle 1{:} \langle m, n \rangle, (trans \circ 2){:} \langle m, n \rangle \rangle =$ (Composition)
$\langle 1{:} \langle m, n \rangle, trans{:}(2{:} \langle m, n \rangle) \rangle =$ (Selection, twice)
$\langle m, trans{:}n \rangle$

In the first step each of the component functions in the construction ("1" and "trans∘2" , respectively) is combined with the total operand $\langle m , n \rangle$ --a sequence of two matrices-- from which in the last step (Selection) each extracts the half it really needs. If the matrices m and n are sizeable , a naive implementation that first copies those matrices and then kicks out half of it again seems absolutely unacceptable on <u>any</u> machine --von Neumann or not-- . The question should be raised what we have achieved. Have we done more than creating a new environment for optimizing compilers? (If that optimization task were well-understood, such a goal could be defended.) The first impression that his functional programming style invites implementations with a lot of concurrency should be complemented by the remark that it invites a lot of traffic that the von Neumann machine doesn't need. Backus's claim that his program "does not name its arguments" is a bit silly, as he distinguishes explicitly between them by ordinal number. (Why didn't the ALGOL Committee make it a language rule that formal parameters would always be identified by "par1, par2, par3, ..." etc. ?)

In the next section "The Algebra of Programs for FP Systems" Backus repeats himself --even more sickeningly: in section 9 he did not go further than referring to "real" programming languages (I was reminded of the general who stated that "NATO was obviously not interested in artificially simplified languages such as PASCAL"), here we even meet our old friend "the average programmer"-- I quote (rather fully in order to show how repetitive Backus writes):

> "So far, proving a program correct requires knowledge of some moderately heavy topics in mathematics and logic [...]. But its theoretical level places it beyond the scope of most amateurs who work outside this specialized field.
>
> If the average programmer is to prove his programs correct, he will need much simpler techniques than those the professionals have so far put forward. The algebra of programs below may be one starting point for such a proof discipline and, coupled with current work on algebraic manipulation, it may also help provide a basis for automating some of that discipline.
>
> One advantage of this algebra over other proof techniques is that the programmer can use his programming language as the language for deriving proofs, rather than having to state proofs in a separate logical system that merely [sic!] talks <u>about</u> his programs."

The section does not present the type of mathematics that I have learned to appreciate for its effectiveness, as the proofs contain a lot of repetitions of the same formulae or parts thereof, and are not free from laboriously belabouring the obvious. He concludes one subsection with:

> "This example (by J.H.Morris, Jr.) is treated more elegantly in [Manna, Z., Ness, S., and Vuillemin, J. Inductive methods for proving properties of programs. Comm.ACM, 16, 8 (Aug. 1973) 491 - 502] on p.498. However, some may find that the above treatment is more constructive, leads one more mechanically to the key questions, and provides more insight into the behavior of the two functions."

The use of the expression "the behavior of functions" is telling.

In the next section "Formal Systems for Functional Programming" are presented, the change being that objects can be used to represent functions.

The notation gets more complicated; typically here Greek letters enter the game. In the last section "Applicative State Transition Systems" Backus seems to try to have his cake and eat it. The example was more elaborate than I wished to cope with and seemed to me a good starter for a 1,000 page manual.

From his "Remarks About Computer Design" I quote:

> "There are numerous indications that the applicative style of programming can become more powerful [sic!] than the von Neumann style."

He may be right, but where are those numerous indications? I did not get them from his article.

In short, the article is a progress report on a valid research effort but suffers badly from aggressive overselling of its significance, long before convincing results have been reached. This is the more regrettable as it has been published by way of Turing Award Lecture.

[1] Backus, J., Can Programming Be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs., Comm.ACM, 21, 8 (Aug. 1978) 613 - 641

[2] Rem, M., Associons and the Closure Statement, Ph.D. Thesis, Eindhoven University of Technology, 12 October 1976

Plataanstraat 5
5671 AL NUENEN
The Netherlands

prof.dr.Edsger W.Dijkstra
Burroughs Research Fellow