Date: April 19, 1972
(Div. & location
S. mail address): San Jose Research Laboratory
Dept. & Bldg: K08/282
line & Tel. Ext.: 7220

IBM

Subject: "Reduction Languages and Variable-free Programming"

Reference:

To:

Distribution List

I am sending you the report "Reduction Languages and Variable-free Programming" because I think it forms the basis for two potential developments of significance to IBM:

1. A well-defined, simple programming language with exceptional definitional capabilities.

2. A radically new kind of computer which could fully utilize the potential power of LSI circuitry, a computer in which all operations are performed on the store, thereby making it the central unit of the machine instead of the present passive device controlled by the CPU.

Now we must look at the structure of this new powerful store!

The subject report describes an entirely new kind of programming language and a new method of programming. These new methods provide great definitional power without requiring any changes in syntax.

Heretofore, there have been two basic types of programming language:

a. Conventional [e.g., FORTRAN, ALGOL, COBOL, APL]

b. Lambda-notation-based, LN languages [e.g., LISP, Applicative Expressions, McG]

Conventional languages offer very limited definitional power without extensive revisions of their syntax and semantics [e.g., if ALGOL lacked its for statement, it could not be introduced as a procedure].

even considering its call-by-name parameter mechanism? [Remember, Jensen's Device]

LN languages have greater definitional power but they are difficult to use for three reasons:

a. They require the use of high-level abstractions [such as function-valued functions] for even simple definitions.

* So does FP! (And all recursive definition schemes?)

b. There are certain basic complexities in their interpretation concerned with the "confusion" and "collision" of bound variables.

The Algol-like languages seem well-received despite being subject to this criticism.

c. Being designed to state function definitions only, they lack the program-like qualities of other languages [e.g., "do this, then this", etc.]

But Red shares the fundamental lack of a notion of state, i.e., ant side-effect, without which the order of evaluation is [virtually] irrelevant.

Therefore, the current situation in programming languages is: LN languages are simpler and have good definitional power but at the cost of having to use high-level abstractions, problems with bound variables, and non-program-like qualities. These problems have caused the rejection of LN languages for general use. Conventional languages continue to be plagued by the problems of complexity and limited definitional power which severely restrict the range of applicability of any one language. These are among the most serious and costly problems in programming today.

[other] LN problems - inefficiency due to:
- retention vs stack
- copy semantics
- no control/data flow!

In contrast, the Red languages of the subject report are simpler in both syntax and semantics than even LN languages; they have exceptional definitional power without the use of high-level abstractions; they have no problems with variables; and they have program-like qualities. In addition, the structure of "programs" [operators] and "data" is the same, hence it is a simple matter for a "program" to operate on itself or on another. This property is unique among current languages. No, Lisp 1.5 also has it.

???

The above properties of Red languages comprise the reasons for claim (1) above, that they could form the basis for the development of simple, well-defined programming languages with unusually powerful definitional capabilities [and the consequent wide range of applicability]. Red languages achieve these advantages by eliminating variables. Their elimination requires the use of unfamiliar variable-free programming techniques. It is not yet certain that the definitional power of Red languages requires the elimination of variables, but if so, then it will be necessary to pay the price of learning these new techniques.

In any case, the absence of variables is really the basis for claim (2) above, that Red languages can form the basis for the development of a radically new kind of computer. Since Red languages use no variables, each "program" must operate on all the information at hand. In machine-like terms: every step of a program has the entire store for its operand.

In present computers, the store is a passive device to which an "address" is sent and from which a small piece of data is then received. All transformations of data take place in the CPU. In the new kind of computer the store is the central device. It is capable of storing complexly structured data with a choice of several representations of its structure. Operations of the sort suggested in the section on Red/3 languages are performed on this store. These operations usually refer to large blocks of data and often involve no arithmetic, even in numerical calculations; instead they change the structure of the data. With the proper representations of data structure, many rearrangements could be performed cheaply. When, however, arithmetic is performed, it is usually applied to large blocks of data and the required accessing of the data is made simpler.

It is still an open question whether such new computers are required for the efficient execution of Red-like programs, for it may be possible to execute them efficiently on current computers in a manner similar to that considered for APL.

In this memorandum, I have tried only to point out the applications of the subject report of clear potential value to IBM [these are not discussed in the report itself]; but I believe there are some contributions to a general theory of programming which may also prove to be useful. The contents of the paper are summarized in its introduction.

John Backus

/pjc