

## SUGGESTION FOR A "MAPPED" EXTENSION OF APL

C. Leibovitz  
University of Alberta  
Computing Center

Users of APL are under the "spell" of beauty, conciseness and elegance of the language. They have however to go back to Fortran, for instance, whenever they cannot avoid a loop executed a great number of times in order to limit the CPU time used.

The natural desire for enlarging the class[1] of cases in which "it would pay" to use APL, is the origin of a great number of suggestions for modifications to and extensions of APL.

However, an APL interpreter is a complicated collection of interrelated software forming a

unity that should not be disrupted. A modification in any part of the collection will have repercussions on the operation of the remainder of the programs and there is no a priori reason preventing these repercussions from being harmful and in need of necessary corrections that may not be welcomed (if at all possible).

It is therefore not enough to show that a given modification is needed; it is necessary to show that it is indeed implementable and has no disruptive character.

Our proposed modification is, in a sense, a "mapping" of an actual interpreter. The logical structure of the mapping is such that we may conclude that: "if there exists an APL interpreter that works, then our mapping will work too."

### Review Of A Non-Modified APL Interpreter

Each time a line is entered from a terminal, the interpreter checks the nature of the line: is it for instance a command? or a line in definition mode? or in execution mode?... Let us designate by CHECK the module of the interpreter that finds out the nature of a line and decides what other module is to handle the line. If the line has been entered in the execution mode, it will be executed from right to left. However, for a number of reasons, this cannot be a straightforward procedure:

1. There is no one-to-one correspondence between a "primitive mathematical symbol" and an execution routine. One same symbol may be a monadic function or may be a dyadic one.
2. The mathematical meaning of a symbol may depend on the nature of another symbol placed at its left.
3. There may be brackets altering the normal right-to-left order of operations.
4. There may be mistakes in the line making it unexecutable.

There must therefore exist a module that will analyse the line, will call execution routines in a proper order and provide those routines with the values of the variables.

We are not concerned here with the way in which this is done, it is enough for us to know

advance. The execution of the function thus becomes faster because there is no need for syntax checking time.

Mapped Level

We recall that CHECK examines the nature of a line and delivers it to GRAM if the line is in execution mode. CHECK has of course other alternatives than calling GRAM. We will not modify the existing alternatives; we will add one alternative more that we call Mapped Level.

The built-in errors can be detected during the mapping operation by MAPGRAM in exactly the same way as GRAM is doing it i.e. by taking over in MAPGRAM the procedure followed by GRAM in