Correspondence

To the Editor
The Computer Journal

Sir

I was interested to read the article 'A Structured Language for Translator Construction' by J. R. White and L. Presser in the February 1975 edition of *The Computer Journal*. There are a number of systems in existence designed to aid the construction of compilers and the like, which use some codification of BNF to drive a syntax analysis operation. One problem which they all have in common is the communication of information between the syntax analysis and the semantic phases. The usual solutions are either to allow semantic procedures to reference syntax definitions and arrange that the system automatically calls these when the associated syntax is recognised, or to allow calls to semantic procedures to be inserted in the textual expression of the BNF, which cause the semantic procedures to be activated when an analysis scan of the syntax reaches a specific point. The system described by White and Presser falls into the former category, the means of reference being the numerical order of the syntax definitions. Another system in this category is the famous Compiler Compiler of Brooker and Morris, where the reference between syntax and semantics is achieved by associating the names of syntactic Phrases and semantic Routines. There seem to me to be two main disadvantages of this approach: firstly, in order to communicate other necessary information from syntax analysis to the semantic process, specifically the particular text which has matched various elements occurring in the syntax definition, a general structure has to be maintained (a stack of information in the case of JOSSLE and an analysis tree in the Compiler Compiler) which may be much larger than is required for the communication, and which has to be understood by the writer of the semantic procedures.

The second disadvantage is that there is no obvious way in which the semantic procedures can influence the course of the syntax analysis. At first sight this seems quite contrary to what is required. However, in most compiler systems constructed in this way, quite a lot of the 'semantic' action is in fact used to perform syntax analysis, simply because there are many rules of syntax which cannot reasonably be expressed in BNF: for example, that all identifiers must be declared, or that only identifiers of a certain type may be used in a given context. In ALGOL 60 a procedure identifier or an array identifier is defined simply as an identifier. In a coded form of the BNF definitions of these it is useful to be able to call a 'semantic' procedure which causes the syntax scan to fail if the identifier is found to be of the wrong type.

The second approach, that of including specific activations of semantic processes in the definitions of the syntax, is used by BCL, the systems programming language designed by D. F. Hendry at the erstwhile University of London Institute of Computer Science, among others. In fact BCL goes much further and embeds all the semantic statements in the syntax definitions, without making any distinction between the two, so that one may write a 'syntax definition' (called a Group) which has a purely semantic action. This avoids the two disadvantages of the first approach, and gives the writer great flexibility. A number of very compact compilers have been written using this system as a result. However, the resulting 'code' comprising the compiler is largely unstructured since it is all embedded in a sequence of definitions which are in fact an unstructured linear sequence just like the usual BNF definition of a language. This is not conducive to using structured programming in the semantic processes. This could be avoided either by textually separating the syntactic and semantic phases, so that semantic actions are initiated only by procedure calls embedded in the syntax definitions, or by imposing some kind of block structure and rules of scope on the definitions in BCL. Textually separating the syntax definitions from the semantic procedures has the advantage that the latter can be written in a structured language (such as JOSSLE) but has the disadvantages that there is less flexibility and that some inefficiency is likely to result from a lot of very short semantic procedures. To impose a block structure and rules of scope on syntax definitions written in BNF or a format based on BNF is an idea which I have not seen discussed anywhere before and might be worth investigating.

However, the main point I wish to make concerning compiler-building systems of this sort is that they are all orientated towards single-pass compilers. This is because the main area which is simplified is the syntax analysis phase, and with a two or more pass compiler, the second and subsequent passes, and part of the first pass all tend to be treated as a semantic phase of the whole process. In fact, with a two pass compiler the intermediate information which is generated by the first pass is 'analysed' by the second pass. However unmodified BNF is not a very suitable means of specifying this analysis because textual strings may not be the best vehicle for the intermediate information, and because there may be more than one stream of information which may not be read in a simple serial manner. For example a symbol table may be accessed randomly, but the records within it may be usefully described by a BNF type of definition. It should be reasonably easy to produce a modified BNF in which the terminals can include other data elements besides character strings and which can include directives to redefine the source of the information to be analysed.

My instinct is to believe that, just as the structure of the information input by a compiler can be defined by declarative syntax definitions (BNF) which can drive an analyser, so should the structure of the information output by a compiler (namely the object code and intermediate information in the case of two pass compilers) be defined by similar syntax definitions which could drive a synthesiser. The relationship between these two would probably have to be described by a third set of statements. The analyser and synthesiser might be regarded as co-operating processes waiting upon events occurring in each other. This notion is supported by the fact that, as I indicated earlier, in some existing systems the syntax definitions refer to the semantic procedures, and in others the semantic procedures refer to the syntax definitions, which suggests that there is no inherent heirarchical relationship between the two.

I would be the first to admit that these ideas, especially those in the last paragraph, are very unformed, but I would be very interested to hear if any work is being done in these directions for I believe the next important practical steps forward in compiler design and construction may lie along these lines. Otherwise, may I hope that my words might provide food for thought for someone with more time and opportunity to pursue these lines of enquiry than myself.

Yours faithfully,

B. T. DENVIR

Standard Telecommunication Research Limited
London Road
Harlow
Essex CM17 9NA
26 March 1975

To the Editor
The Computer Journal

Sir

Decimal number checking schemes

Andrew (1974) mentions the problem of finding a scheme to attach two decimal digits to a two-place decimal number so as to allow correction of single errors. This may be viewed as the problem of finding a mapping from the set $\{00, 01, \ldots, 99\}$ into itself (the image being the check digits) such that the resulting set of code words is single-error-correcting. It is well-known that a necessary and sufficient condition for a set of code words to be single-error-correcting is that the Hamming distance between any two code words should be at least three.