transfer to the appropriate state in the arithmetic expression processor. If Resume error returns, these states also given an error return.

In addition to syntax checking, the program has been designed to do some simple error correction. State AREX6 looks for a specific error, the use of a conditional expression following a **then**. When **if** is found at this point, a left parenthesis is inserted ahead of the **if** and the arithmetic expression processor is called recursively to process a complete conditional arithmetic expression. On a normal return from this call, a right parenthesis is inserted and processing continues. Any such corrections are accompanied by a message to the user indicating what action has been taken.

The method of syntax checking described in this paper has been implemented and tested on the IBM 7090 computer and is now being used as the syntax checking phase of the SHARE ALGOL 60 Translator. In general, the results have been very satisfactory: most errors are detected in one machine run; syntactically correct programs are checked very rapidly, and no time is wasted in attempting to translate syntactically incorrect programs.

## *PRACNIQUE*

# A NOTE ON THE FORMATION OF FREE LIST

The concept of an available-space list was introduced by Newell and Shaw [1] in 1957, and has since been incorporated into a number of different systems [2–5]. The *available-space list* (or "free list") is a list of all available memory locations. It should initially be as large as possible, and ideally it would contain every cell not used by the program. The subject of this note is the initial formation of a free list on the IBM 7090-7094, using the FORTRAN II monitor, version 2. The method presented originated while the authors were working on an implementation of the WISP [5] system for the 7090 in cooperation with Prof. M. V. Wilkes and his colleagues.

One method of obtaining a free list, proposed by Weizenbaum [2], is to use DIMENSION and COMMON statements to define it as an array. This method will generally produce a free list which does not utilize all of the available space, or which causes an overlap between program and COMMON storage areas. The specification of the length of the free list can be avoided by the use of information provided by the loader at object time. Under FORTRAN II, the program break (first location not used by any program) and the COMMON break (first location below the COMMON storage area) are stored in the decrement and address fields, respectively, of word 143 (octal). Thus the programmer has access to the limits of available core at object time. The routine in Fig. 1 will organize this space into a one-way list structure whose pointer is in the COMMON location FREE.

```
        CLA     LIMITS      GET PROGRAM BREAK,
                              COMMON BREAK
        STA     FREE        COMMON BREAK =
                              FIRST ELEMENT OF
                              LIST
        STD     TEST        PROGRAM BREAK =
                              LAST ELEMENT OF
                              LIST
        PAX      ,4
LOOP    SXA     *+2,4       ADDRESS OF PRESENT
                              ELEMENT
        TXI     *+1,4,-1    GET ADDRESS OF NEXT
                              ELEMENT
        SXA     **,4        PLACE IN ADDRESS
                              FIELD OF PRESENT
                              ELEMENT
TEST    TXH     LOOP,4,**
LIMITS  BOOL    143
FREE    COM-    1
          MON
```

FIG. 1

In this list, the address field of each element except the last contains the address of the next element. If the routine is used immediately after loading, all fields other than the address will contain zeros, as will the address field of the last element. If the routine is executed later in the course of the program, it may be necessary to add the instruction

STZ* TEST −1

immediately following LOOP to insure that the cells are cleared.

Other list formats may be generated by simple modifications of the routine. For example, to produce a SLIP [2] free list, insert the instruction

ADD = 2B17

between the second and third instructions above, and change LOOP +1 to

TXI *+1,4,−2


REFERENCES:

1. NEWELL, A., AND SHAW, J. C. Programming the logic theory machine. Proc. Western Joint Comput. Conf. 1957, 230–240.
2. WEIZENBAUM, J. Symmetric list processor. *Comm. ACM 6* (Sept. 1963), 524–544.
3. MCCARTHY, J. Recursive functions of symbolic expressions and their computation by machine, part I. *Comm. ACM 3* (Apr. 1960), 184–195.
4. GELERNTER, H., ET AL. A FORTRAN-compiler list-processing language. *J. ACM 7* (Apr. 1960).
5. WILKES, M. V. An experiment with a self-compiling compiler for a simple list-processing language. In *Annual Review of Programming*, *V. 4*, Goodman (Ed.), (1963).



WILLIAM M. WAITE
H. SCHORR
*Columbia University*
*New York, N. Y.*



RECEIVED FEBRUARY, 1964