ALGOL

for the

LGP - 30

A Comparison

Computation Center
Dartmouth College
Hanover, New Hampshire
15 February 1962
ALGOL FOR THE LGP-30

INTRODUCTION

This report describes the ALGOL system prepared for the Royal McBee LGP-30 by the Dartmouth College Computation Center. The description is made with reference to the "Report on the Algorithmic Language ALGOL 60" (Communications of the ACM, May 1960) and assumes a knowledge of that language. This report is not an ALGOL primer, no does it contain specific instructions of the use of the ALGOL system described. Instead, details of grammar are included to permit a critical evaluation of this effort to squeeze a subset of ALGOL into a small computer.

For verbal convenience, the ALGOL system herein described will be referred to as ALGOL-30, a contraction of the title of this report.

The report is divided into four parts: First, a brief historical background is given together with well-deserved acknowledgments. Second, a brief description of the ALGOL-30 system is given. Third, the grammar of ALGOL-30 is compared paragraph by paragraph with the Report. Fourth, ALGOL-30 is compared with SMALGOL, a recently formalized subset of ALGOL.

HISTORICAL BACKGROUND

Our original purpose in devising the ALGOL-30 system was to provide a reasonable algebraic language system for the LGP-30. In the meantime, several excellent and efficient algebraic language systems have been developed for that machine, and it was decided not to try to compete with them in terms of speed and efficiency. Rather, it was felt desirable to try to incorporate as much of ALGOL as possible even if it meant sacrificing run-time efficiency. Even so, it has turned out that the compiling times are exceptionally small and that running speed for arithmetic operations is quite fast. The system's speed suffers when handling matrix elements, procedure calls, variables called by name, and, to a certain extent, "for" statements. Nonetheless, it is expected that the ALGOL-30 system will see extensive use in a small number of LGP-30 installations.

It should be recorded that this unusually fine accomplishment is due almost entirely to the efforts of four Dartmouth undergraduate research assistants: Stephen J. Garland, Robert F. Hargraves, Anthony W. Knapp, and Jorge Llacer. Hargraves and Llacer graduated before the system could be completed, but their contribution during the initial stages of the work was invaluable. It should be mentioned that our becoming involved in this project was a direct
result of Harpens' having devised a complete algebraic language system (DART) during the summer of 1959. The four students have done all of the planning, coding, and debugging, the role of this report's author being merely that of arbiter of the restrictions and interpretations of ALGOL.

BRIEF DESCRIPTION

The common denominator of the ALGOL-30 thinking was to produce a system as close in detail and philosophy to ALGOL-60 as possible. Features not ordinarily found in small computer compiler systems were included. For instance, procedures may be compiled separately and linked together at run time, a feature that requires the translator to produce code with non-absolute addresses.

The system consists of three parts

1. Translator
2. Loader
3. Interpreter.

The output of the translator is semisymbolic code having a 64 instruction repertory and symbolic addresses. The loader assigns absolute addresses, data storage, and links procedures. The interpreter performs the arithmetic and other operations necessary to carrying out the program.

The translator uses the last in, first out principle discussed in detail by Samuelson and Bauer (February 1960 Communications.) It was found possible to plan the entire translator on that basis, not only arithmetic expressions. The symbol table look up follows the suggestion of Williams (June 1959 Communications) and was found to be exceptionally fast without requiring excessive storage. The THUNK idea of Irons and Feuerzeig (January 1961 Communications) is used to implement procedures.

The loader assigns addresses and links procedures. If a program has been previously assembled, it may be dumped in hexadecimal (the nearest thing to absolute machine language in the LGP-30 and similar to the octal or binary mode on other computers) and hexadecimal programs can also be loaded.

The interpreter performs the real arithmetic that is not a part of the LGP-30 hardware. This type of run time package was selected to lessen the amount of code required to be produced by the translator. For instance, for $d_1 = a + b + c$ the translator generates one bring, two adds, and one hold, or four instructions in all. The closed sub-routine approach would require a total of nine instructions for the same statement. The interpreter approach thus requires less space for programs but is slightly slower because of the additional interpret time.

The interpreter contains subtract-from and divide-into sequences, plus a number of double purpose instructions that lessen the work of the translator and permit including more features of ALGOL.

A trace mode may be invoked any time during the running of a
program. Printed will be all statement labels, and all variables appearing in the left part of an assignment statement together with the value assigned, all in source program symbolism.

COMPARISON WITH ALGOL 60

In the following comparison, the paragraph numbers refer to the Report. Included here are only those paragraphs for which ALGOL-30 has been restricted.

2.1 Letters - only one case of letters can be distinguished because the standard LGP-30 cannot read the case shift characters.

2.2.1 Digits - All digits are allowed except that "one" and the letter "el" are the same.

2.2.2 Logical Values - No change. (However, it is noted that each ALGOL-30 symbol composed of letters does have the obvious relation to the letters from which they are composed; this in contrast with footnote of the Report.)

2.3 Delimiters - The following is a list of the ALGOL-30 delimiters that differ from those of ALGOL

<table>
<thead>
<tr>
<th>ALGOL</th>
<th>ALGOL-30</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>*/</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>\</td>
<td>\</td>
</tr>
<tr>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>!=</td>
<td>!=</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&lt;=</td>
</tr>
<tr>
<td>&gt;=</td>
<td>&gt;=</td>
</tr>
<tr>
<td>equiv</td>
<td>equiv</td>
</tr>
<tr>
<td>imply</td>
<td>imply</td>
</tr>
<tr>
<td>or</td>
<td>or</td>
</tr>
<tr>
<td>and</td>
<td>and</td>
</tr>
<tr>
<td>not</td>
<td>not</td>
</tr>
<tr>
<td>go to</td>
<td>go to or goto</td>
</tr>
<tr>
<td>;</td>
<td>;</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>\</td>
<td>(strings not included)</td>
</tr>
<tr>
<td>string</td>
<td>(not included)</td>
</tr>
<tr>
<td>;</td>
<td>(not included)</td>
</tr>
<tr>
<td>;</td>
<td>(not included)</td>
</tr>
</tbody>
</table>

In ALGOL-30 spaces are not scanned for by the translator. Hence, "a b" is different from "ab". Because of hardware considerations, it was found convenient to have each delimiter, label, and identifier set off by a symbol separator (ʼ). Furthermore only the last five characters of a given word, symbol or delimiter are entered. Thus, "switch" and "witch" are equivalent. Although the upper and lower case and the back space keys cannot be inputted, they may be used ad-lib to improve
the external copy, although this practice is not to be recommended. Thus, ::= the symbol for : could be typed :, backspace, :, and only : would show on the copy. The same is true for ,.

2.4 Identifiers - No restriction except that noted above, that only the last five letters are used.

2.5 Numbers - Full freedom on the expression of numbers is permitted except that multiple use of some of the number keys on the flexwriter requires numbers in ALGOL-30 to be set off by a double symbol delimiter (""). Type is correctly assigned in all cases. All the examples in section 2.5.2 are permitted except that the symbol separator (!) must appear at least every five characters. Thus "-200.084" is permissible, but "+200.084" will be translated as +200.084 since the (+) sign would be lost.

2.6 Strings - Strings are not included.

3.3 Arithmetic Expressions. No change in syntax, including the possibility of mixing conditionals in any way. ALGOL-30 permits any type of mixed expression according to 3.3.4.1 and 3.3.4.2. However, certain expressions involving ++ can confuse ALGOL-30. a++b, a and b integer, is defined to be of type integer if b > 0 but of type real is b<0.

If a, b and c are integers, then the system will compute a = b + a correctly if b > 0, and will generate an error stop otherwise. However, if x is of type real, then both c = x + a++b and x = a++b will be correctly computed even if b<0.

3.3.5 Precedence of operators - Because of the nature of the translator, some expressions are not evaluated from left to right. All other rules of precedence and parentheses are followed, including evaluating subscript expressions for left part variables before evaluating the right part expression.

3.3.6 Real arithmetic - Real arithmetic is performed in floating point having approximately seven significant figures. Of course, the nature of the arithmetic may be changed without affecting the syntax by using a different interpreter package.

3.4 Boolean expressions - Boolean expressions are permitted in complete detail in accord with paragraphs 3.4, 3.4.2, 3.4.3, 3.4.5. However as with arithmetic expressions, evaluation may not always be from left to right as requires by 3.4.6.

3.5 Designational Expressions - No change. Labels may be unsigned integers or identifiers, and designational expressions are permitted, however, leading zeros are not ignored as required by 3.3.5.
4.7 Procedures - Procedures have been implemented with full freedom for calling variables by name or by value. In particular, the pathological example discussed by Arden, Galler and Graham in the July 1961 Communications can be run. However, because of storage allocation problems, "own arrays" and "arrays" called by "value" have not been included.

5.2 Arrays declarations. "Own arrays" are not permitted.

5.2.4 Lower upper bound expressions - Since the ALGOL-30 system contains no run-time executive routine, lower upper bounds must always be constants.

COMPARISON WITH SMALGOL

The source for the comparison of ALGOL-30 with SMALGOL is the article "SMALGOL-61" in the November 1961 Communications. In general, the languages are very similar with SMALGOL being slightly more restrictive. The comparison will be by paragraph number, mentioning below only those paragraphs where there is a difference.

2.2.1 Digits. The digit "1" and the letter lower case "l" are the same in ALGOL-30.

2.2.2 Logical values. Logical values are permitted in ALGOL-30.

2.3 Delimiters. Certain of the delimiters of SMALGOL have a different representation in ALGOL-30. Also, the left and the right quotes are not in ALGOL-30. ALGOL-30 does not permit inter-changing "[" and "]" with "(" and ")". Spaces have a letter interpretation in ALGOL-30 since the apostrophe is used as the symbol delimiter.

3.2 Function designators. ABS and SIGN may be used as procedure parameters.

3.3 Arithmetic Expressions. a+b is correctly interpreted in most cases (except as noted in 3.3 earlier.)

3.4 Boolean expressions. These are allowed in ALGOL-30 with no restrictions.

3.5 Designational Expressions. Unsigned integers may be used as labels.

4.2 Assignment Statements. Boolean assignment statements are allowed in ALGOL-30.

4.6 "For" statements. The complete generality of "for" statements is included in ALGOL-30 even though less efficient routines will result. The "for" list may include any
number of the three types of elements in any order. Of course, the repeated statement is treated as a subroutine.

4.7 Procedure statements. No strings are allowed in ALGOL-30 although labelling of data at translate time or run-time will be provided for. Procedures may be declared at any time since the linking loader operates just before run-time. ALGOL-30 does not require the parameters of a procedure which itself is a parameter of another procedure to be called by value. In fact, procedures may be nested to any depth with full freedom for calling parameters by name or by value, including ABS and SIGN.

5.1 Type declarations. Boolean variables may be declared.

5.3 Switch declarations. Full Freedom for switch lists is permitted.

5.4 Procedure declarations. Procedures that define values may have parameters called by name.

There may be slight errors or omissions in the above comparison, as there certainly also are in the previous comparison with ALGOL. However, in broad outline ALGOL-30 compares favorably with SMALGOL, being less restrictive in the handling of Boolean variables, for lists, and procedure parameters, and being more restrictive in no essential way.