R. A. Kelley

APLGOL, an Experimental Structured Programming Language

Abstract: An experimental programming language called APLGOL adds structured programming facilities to the existing framework of APL. The conventional semantics of APL is unaltered and only minor changes are incorporated in the syntax. The advantages of the proposed interstatement structuring and control are outlined.

Programs designed and written using "structured" programming techniques have been demonstrated to be more readily produced, more reliable, and more easily maintained than unstructured programs [1]. These techniques essentially involve arranging the application into principal components, which in turn, are further organized to produce a set of highly structured procedures. For this purpose, key programming language constructs have been used in conjunction with many languages to highlight interstatement structuring and control.

Since structured programming techniques have been successful when applied to programming in other languages, they should be equally advantageous for APL programming efforts. In APL the compact and concise operators extend to vector or array operands, and single expressions often can subsume the equivalent of several statements in a language such as ALGOL or PL/1. However, even in spite of the famous APL "one-liners", many APL programs do require quite a few statements and frequently utilize rather complicated control flow. If this were highlighted by structured programming language constructs, the resulting programs should prove easier to write and debug, and more importantly, easier for others to read and understand.

Consequently, the APLGOL language [2], based on a notation of Abrams [3], is an experiment to add structured programming language constructs to the basic APL framework. In adding the structured statements, a deliberate effort has been made to augment APL in the areas where these constructs are absent, and to avoid establishing other constructs, e.g., a new rule for name scope, that would tend to compete with existing APL facilities. Close attention has been paid to the relation between the new APLGOL facilities and existing APL operations. In this sense, APL was considered the target machine language to which APLGOL source programs could be compiled readily without materially affecting the speed or size of the object program or the compiler, or otherwise distorting the APL system.

APLGOL semantics

Before any new features were incorporated in APLGOL, it was decided first that all the semantic functions of APL should appear in APLGOL without change, since removing or altering any of them would be a step in the wrong direction. To achieve this (and still attain the goal of furnishing structured programming functions), the APL procedure format with its procedural header and numbered statements was abandoned in favor of a free form in which statements could span lines in an arbitrary manner. Thus, a new *PROCEDURE* statement having a different syntax but identical semantics, and a new *END PROCEDURE* statement were incorporated to contain the procedure body.

Also, a slight change was introduced in the original APL statement syntax to permit a semicolon not enclosed in subscript brackets to terminate a statement. The semicolon formerly used to catenate items for printing was replaced with the union symbol, "∪". The syntax for comments was changed to require the comment delimiter to follow as well as to precede the comment. In

this way comments can now be embedded within statements. Otherwise, the syntax of APL was left unaltered, and an APL program written in this new format can be compiled to one identical to the original form.

To this basic APL framework were added new statements for interstatement control, conditional statement execution, and statement structuring, as well as a few statements (such as the *RETURN* statement) to clarify the intent of the algorithm. The most fundamental of these statements was the *IF* statement, optionally having an *ELSE* clause. The following example shows an APLGOL *IF* statement, together with the corresponding APL object text:

```
APLGOL Source          APL Object Text

IF A≠0 THEN              →(~A≠0)/Q001
   B←C÷A;                B←C÷A
ELSE                     →Q002
   D←B+4;                Q001:D←B+4
                         Q002: . . .
```

The conditional expression can be any APL expression producing a scalar 0 or 1. The true part and the optional false part are compiled into separate APL statements to be executed depending on the condition. This feature is, of course, defined recursively to permit the nesting of *IF* statements to any arbitrary depth. In APL this facility can be obtained only by using several statements requiring branches and labels that quickly obscure the intent of the program. The function is far more clearly identifiable in the APLGOL *IF* statement form. Appendix 3 shows examples of APLGOL source statements and the corresponding APL object text for the remaining new statement types.

Additionally, a simple statement block analogous to the *DO-END* statement pair in PL/1 was incorporated into group statements for a common purpose, such as to delimit a block of statements representing either the true or false portion of an *IF* statement. Used with graphic formatting to indent statements common to a block, this simple block facility, together with the *IF* statement, provides much of the necessary structured programming facilities. Other facilities, however, add more convenience for the programmer and help further to highlight the intent of the algorithm.

Several other types of blocks were included for more concise expression of common functions. These included a *WHILE* block, used in both ALGOL and PL/1, a *REPEAT* block similar to the *WHILE*, except that the condition is tested at the end of the block, a *CASE* block [4] for selective execution of one of the statements (or blocks) contained in it, and finally an iteration statement, as in the ALGOL *FOR* statement or the PL/1 iterative *DO* statement, for iterative execution of a statement or block. In these blocks the conditional expressions are permitted to be any APL expression producing a scalar 0 or 1. The index expression in the *CASE* block can be any APL expression producing a positive scalar integer to index one of the statements or blocks contained within it. (The index origin of the target APL interpreter still must be implicitly understood by the programmer.) Finally, any scalar APL expression can be used for the initialization, step, and test in the iteration statement; the latter two are recomputed on each cycle.

To complete the language extensions, a *RETURN* statement and a null statement are incorporated, respectively, to effect a return from the current procedural level and to provide an empty statement for use in conjunction with *CASE* blocks. An optional expression in the *RETURN* statement may be used for any computation prior to returning. This differs from the PL/1 version, in which the expression may only relate to the value to be returned from a function-procedure.

APLGOL syntax

Once the semantic functions had been determined, the question of syntax seemed to be a matter of personal preference. APLGOL semantics essentially can be represented in either an ALGOL or a PL/1 syntax, and the user community could be expected to be familiar with either. Technically, table-driven syntax recognizers could easily make the recognition of either syntax a simple matter; so both an ALGOL-like and a PL/1-like syntax were created (see Appendices 1, 2, respectively). An APLGOL procedure employing either syntax may be written and compiled, but mixtures of the two are not permitted.

Perhaps an example of a standard APL program should now be contrasted with its two APLGOL counterparts (see Figures 1, 2, and 3). The example used is the "deal" operator written in APL, cf. [5].

Because the blocks are enclosed by keywords such as *BEGIN*, *DO*, *WHILE*, *CASE*, *FOR*, and *END*, the programmer is far more aware of the statements which they comprise. These programs are relatively short and uncomplicated, yet the control flow in the conventional APL program is not at all obvious. On the other hand, the blocks in the APLGOL programs are quickly discernible—especially if they are indented as shown. Also, notice that the algorithms in these samples are basically an *IF . . . THEN . . . ELSE* statement.

The APLGOL compiler

The same APLGOL compiler accommodates either syntax by selecting the appropriate set of syntax tables. It generates APL object text in one pass with an additional pass over the object text to remove extraneous labels and branches resulting from nested blocks or user-defined labels.

Figure 1 The unstructured APL version.

```
    ∇Z←A DEAL B;I;J
[1]   ⍝GLOBAL VARIABLES:Q
[2]   ⍝USES:ROLL
[3]   'RANK' ERROR 1≠⍴⍴/⍴A
[4]   'RANK' ERROR 1≠⍴⍴/⍴B
[5]   'DOMAIN' ERROR 0≠TYPE A
[6]   'DOMAIN' ERROR (A<0)∨A≠⌊A
[7]   'DOMAIN' ERROR 0≠TYPE B
[8]   'DOMAIN' ERROR B≠⌊B
[9]   'DOMAIN' ERROR A>B
[10]  →SHORT IF A<⌊B÷16
[11]  Z←(Q-1)+⍳B
[12]  →END IF A=0
[13]  I←0
[14] LOOP:J←1+I+(ROLL B-I)-Q
[15]  I←I+1
[16]  Z[I,J]←Z[J,I]
[17]  →LOOP IF A>I
[18] END:Z←A↑Z
[19]  →0
[20] SHORT:Z←⍳0
[21] OUTER:→0 IF A=⍴Z
[22] INNER:I←ROLL B
[23]  →INNER IF I∈R
[24]  Z←Z,I
[25]  →OUTER
    ∇
```

Figure 2 APLGOL version using ALGOL-like syntax.

```
PROCEDURE Z←A DEAL B,I,J;
   ⍝ GLOBAL VARIABLES: Q ⍝
   ⍝ USES: ROLL ⍝
   'RANK' ERROR 1≠⍴⍴/⍴A; 'RANK' ERROR 1≠⍴⍴/⍴B;
   'DOMAIN' ERROR 0≠TYPE A; 'DOMAIN' ERROR (A<0)∨A≠⌊A;
   'DOMAIN' ERROR 0≠TYPE B; 'DOMAIN' ERROR B≠⌊B;
   'DOMAIN' ERROR A>B;
   IF A≥⌊B÷16 THEN
      BEGIN
         Z←(Q-1)+⍳B;
         IF A=0 THEN RETURN Z←A↑Z;
         I←0;
         REPEAT
            J←1+I+(ROLL B-I)-Q; I←I+1; Z[I,J]←Z[J,I];
         UNTIL A≤I;
         RETURN Z←A↑Z;
      END
   ELSE
      BEGIN
         Z←⍳0;
         WHILE A≠⍴Z DO
            REPEAT I←ROLL B; UNTIL ~I∈R;
            Z←Z,I;
         END
      END
END PROCEDURE
```

The entire compiler, * written in APL (and APLGOL) comprises 12 procedures containing a total of 250 APL object statements. It was completed in well under one man-month and can compile approximately 100 APL object statements per CPU minute of IBM System/360-75 time, using less than a 30,000-byte workspace.

Discussion

The facilities incorporated in APLGOL have already proven useful in the context of other less powerful programming languages. However, the basic power in APL tends to lie at the expression level, where elegant operators can be associated with scalar, vector, or array data to perform functions requiring quite a few statements in other languages. The new facilities in APLGOL are really concerned with interstatement control structure, which is weak in APL. Thus it would appear that the expression power of APL is complemented by these interstatement structuring facilities.

The APLGOL compiler itself still remains among the largest examples of APLGOL programming. It has been used as a model of structured programming and has served as an excellent aid in teaching beginners the intricacies of compiler design. Of course it is not an exceedingly complex compiler, but the structured algorithms clearly permit one rapidly to discern its overall framework and to examine the various sections in greater detail.

*The APLGOL compiler was written as an experimental project and is not available for external distribution.

Figure 3 APLGOL version using PL/1-like syntax.

```
PROCEDURE Z←A DEAL B,I,J;
   ⍝ GLOBAL VARIABLES: Q ⍝
   ⍝ USES: ROLL ⍝
   'RANK' ERROR 1≠⍴⍴/⍴A; 'RANK' ERROR 1≠⍴⍴/⍴B;
   'DOMAIN' ERROR 0≠TYPE A; 'DOMAIN' ERROR (A<0)∨A≠⌊A;
   'DOMAIN' ERROR 0≠TYPE B; 'DOMAIN' ERROR B≠⌊B;
   'DOMAIN' ERROR A>B;
   IF A≥⌊B÷16 THEN
      DO;
         Z←(Q-1)+⍳B;
         IF A=0 THEN RETURN Z←A↑Z;
         I←0;
         REPEAT
            J←1+I+(ROLL B-I)-Q; I←I+1; Z[I,J]←Z[J,I];
         UNTIL A≤I;
         RETURN Z←A↑Z;
      END;
   ELSE
      DO;
         Z←⍳0;
         DO WHILE A≠⍴Z;
            REPEAT I←ROLL B; UNTIL ~I∈R;
            Z←Z,I;
         END;
      END;
END PROCEDURE;
```

APLGOL does not represent new programming language technology; it was never intended to do so. Instead, it has served as an experimental means for providing known structured programming language facilities in APL without otherwise destroying the basic APL philosophy. This has not entailed major changes in APL; indeed, the APL semantics remain unaltered, and only minor changes have been made in the APL syntax. The

compiler approach has provided this function without changing the standard APL system.

References

1. F. T. Baker, "Chief Programmer Team Management of Production Programming", *IBM Systems Journal*, **11**, No. 1, 56 (1972).
2. R. A. Kelley, "APLGOL, A Structured Programming Language for APL", *IBM Palo Alto Scientific Center Report* No. 320-3299, August 1972.
3. P. S. Abrams, "An APL Machine", SLAC Report No. 114, Computer Science Department, Stanford University, February 1970.
4. N. Wirth, and C. Hoare, "A Contribution to the Development of ALGOL", *Communications of the ACM*, **9**, No. 6, 413-432 (June 1962).
5. R. H. Lathwell and J. E. Mezei, "A Formal Description of APL", IRIA Colloque APL, pp. 181-215, September 1971.
6. Richard Sites, "ALGOL-W," Stanford Computer Science Report 71-230, February 1972.

Received April 25, 1972

The author is located at the IBM Data Processing Division Scientific Center, 2670 Hanover Street, Palo Alto, California 94304.

Appendix 1. ALGOL-like syntax for APLGOL

```
 1  <PROGRAM> ::= _|_ <PROCEDURE> ; <STATEMENT LIST> END PROCEDURE _|_

 2  <PROCEDURE> ::= PROCEDURE <EXPRESSION>

 3  <STATEMENT LIST> ::= <STATEMENT>
 4                     | <STATEMENT LIST> <STATEMENT>

 5  <STATEMENT> ::= <BASIC STATEMENT>
 6                | <IF STATEMENT>
 7                | <FOR>

 8  <LABELIST> ::= <LABEL> :

 9  <IF STATEMENT> ::= <IF CLAUSE> <STATEMENT>
10                   | <IF CLAUSE> <TRUE PART> <STATEMENT>
11                   | <LABELIST> <IF STATEMENT>

12  <IF CLAUSE> ::= IF <EXPRESSION> THEN

13  <TRUE PART> ::= <BASIC STATEMENT> ELSE

14  <FOR> ::= <FOR MAX> <STATEMENT>
15          | <LABELIST> <FOR>

16  <FOR MAX> ::= <FOR LIMIT> DO
17              | <FOR STEP> DO

18  <FOR STEP> ::= <FOR LIMIT> BY <EXPRESSION>

19  <FOR LIMIT> ::= <FOR HEAD> TO <EXPRESSION>

20  <FOR HEAD> ::= FOR <EXPRESSION>

21  <BASIC STATEMENT> ::= <BASIC STATEMENT-A> ;
22                      | <GROUP> <STATEMENT LIST> END
23                      | ;
24                      | <LABELIST> <BASIC STATEMENT>

25  <BASIC STATEMENT-A> ::= <EXPRESSION>
26                        | RETURN
27                        | RETURN <EXPRESSION>
28                        | REPEAT <STATEMENT LIST> UNTIL <EXPRESSION>
29  <GROUP> ::= BEGIN
30            | <CASE/WHILE>

31  <CASE/WHILE> ::= <CASE HEAD> DO
32                 | <WHILE HEAD> DO

33  <WHILE HEAD> ::= WHILE <EXPRESSION>

34  <CASE HEAD> ::= <CASE HEAD-A> OF <EXPRESSION>

35  <CASE HEAD-A> ::= CASE <EXPRESSION>
```

*The ALGOL cited here is modeled after ALGOL-W from Stanford University.[6]

Appendix 2. PL/1-like syntax for APLGOL

```
 1  <PROGRAM> ::= _|_ <PROCEDURE> ; <STATEMENT LIST> END PROCEDURE ; _|_

 2  <PROCEDURE> ::= PROCEDURE <EXPRESSION>

 3  <STATEMENT LIST> ::= <STATEMENT>
 4                     | <STATEMENT LIST> <STATEMENT>

 5  <STATEMENT> ::= <BASIC STATEMENT>
 6                | <IF STATEMENT>
 7                | <FOR> <END>

 8  <LABELIST> ::= <LABEL> :

 9  <IF STATEMENT> ::= <IF CLAUSE> <STATEMENT>
10                   | <IF CLAUSE> <TRUE PART> <STATEMENT>
11                   | <LABELIST> <IF STATEMENT>

12  <IF CLAUSE> ::= IF <EXPRESSION> THEN

13  <TRUE PART> ::= <BASIC STATEMENT> ELSE

14  <FOR> ::= <FOR MAX> <STATEMENT LIST>
15          | <LABELIST> <FOR>

16  <FOR MAX> ::= <FOR LIMIT> ;
17              | <FOR STEP> ;

18  <FOR STEP> ::= <FOR LIMIT> BY <EXPRESSION>

19  <FOR LIMIT> ::= <FOR HEAD> TO <EXPRESSION>

20  <FOR HEAD> ::= DO <EXPRESSION>

21  <BASIC STATEMENT> ::= <BASIC STATEMENT-A> ;
22                      | <GROUP> <STATEMENT LIST> <END>
23                      | ;
24                      | <LABELIST> <BASIC STATEMENT>

25  <BASIC STATEMENT-A> ::= <EXPRESSION>
26                        | RETURN
27                        | RETURN <EXPRESSION>
28                        | REPEAT <STATEMENT LIST> UNTIL <EXPRESSION>
29  <GROUP> ::= <DO BLOCK>
30            | <CASE/WHILE>

31  <CASE/WHILE> ::= <CASE HEAD> ;
32                 | <WHILE HEAD> ;

33  <WHILE HEAD> ::= <WHILE HEAD-A> <EXPRESSION>

34  <CASE HEAD> ::= <CASE HEAD-A> OF <EXPRESSION>

35  <CASE HEAD-A> ::= DO CASE <EXPRESSION>

36  <WHILE HEAD-A> ::= DO WHILE

37  <DO BLOCK> ::= DO ;

38  <END> ::= END ;
```

Appendix 3. APLGOL statement semantics

1. *IF STATEMENT*

ALGOL SYNTAX	PL/I SYNTAX	APL OBJECT TEXT
IF A≠0 THEN	IF A≠0 THEN	→(~A≠0)/Q001
B←C÷A;	B←C÷A;	B←C÷A
. . .	. . .	Q001: . . .

IF STATEMENT (WITH ELSE)

ALGOL SYNTAX	PL/I SYNTAX	APL OBJECT TEXT
IF A≠0 THEN	IF A≠0 THEN	→(~A≠0)/Q001
B←C÷A;	B←C÷A;	B←C÷A
ELSE	ELSE	→Q002
B←B+4;	B←B+4;	Q001:B←B+4
. . .	. . .	Q002: . . .

2. *WHILE STATEMENT*

ALGOL SYNTAX	PL/I SYNTAX	APL OBJECT TEXT
WHILE A∈B DO	DO WHILE A∈B;	Q001:→(~A∈B)/Q002
. . .	. . .	. . .
END	END;	→Q001
. . .	. . .	Q002: . . .

3. *REPEAT STATEMENT*

ALGOL SYNTAX	*PL/I SYNTAX*	*APL OBJECT TEXT*
REPEAT . . . *UNTIL AϵB;* . . .	*REPEAT* . . . *UNTIL AϵB;* . . .	*Ω001: . . .* . . . *→(~AϵB)/Ω001* . . .

4. *CASE STATEMENT*

ALGOL SYNTAX	*PL/I SYNTAX*	*APL OBJECT TEXT*
CASE BT[I] OF 4 DO *A←B;* *C←D;* *E←F;* *G←H;* *END*	*DO CASE BT[I] OF 4;* *A←B;* *C←D;* *E←F;* *G←H;* *END;*	*→(Ω001,Ω002,Ω003,Ω004)[BT[I]]* *Ω001:A←B* *→Ω005* *Ω002:C←D* *→Ω005* *Ω003:E←F* *→Ω005* *Ω004:G←H* *Ω005: . . .*

5. *ITERATION STATEMENT*

ALGOL SYNTAX	*PL/I SYNTAX*	*APL OBJECT TEXT*
FOR I←J+1 TO ρA DO . . .	*DO I←J+1 TO ρA;* . . . *END;*	*I←(J+1)-1* *Ω001:→((ρA)>I←I+1)/Ω002* . . . *→Ω001* *Ω002: . . .*

6. *RETURN STATEMENT*

ALGOL SYNTAX	*PL/I SYNTAX*	*APL OBJECT TEXT*
RETURN; *RETURN ⍞←'ERROR';*	*RETURN;* *RETURN ⍞←'ERROR';*	*→0* *→0,ρ⍞←'ERROR'*

7. *SIMPLE BLOCK STATEMENT*

ALGOL SYNTAX	*PL/I SYNTAX*	*APL OBJECT TEXT*
BEGIN *A←B+1++/C;* . . . *C←A;* *END*	*DO;* *A←B+1++/C;* . . . *C←A;* *END;*	*A←B+1++/C* . . . *C←A*

8. *NULL STATEMENT*

ALGOL SYNTAX	*PL/I SYNTAX*	*APL OBJECT TEXT*
A←+/B; ; *C←C|A;*	*A←+/B;* ; *C←C|A;*	*A←+/B* *C←C|A*