A NOTE ON THE ALGOL 68 PROGRAMMING

LANGUAGE

D. Andermann.
Technical Report No. 7
June 1970
A Note On The Algol 68 Programming

Language

The following paper is intended as an introduction to the language. It is not a complete description of it, but it is hoped that it does emphasize the important ideas of the language.

The language we intend to present is a new language. It is not an extension of an existing programming language and it contains many good points from existing languages discarding at the same time their bad ones. As is known, Fortran and Algol 60 are scientifically oriented languages, and Cobol as its name shows (Common Business Oriented Language) is oriented towards commercial applications. While working with Fortran and Cobol one meets a first inconvenience, which is the writing in fixed columns. It is very unpleasant to waste several compilations only because one did not write in the proper columns. In Algol 60, this limitation does not exist and similarly it does not exist in Algol 68, where the writing format is free. For example the following is perfectly valid (As was already the case in Algol 60):

```
X := 1;
```
Method of Description of the Language

I don't intend to discuss this topic in detail, because for everyday work it is not so important to know how the language is described, but as it is a new method of description I think it should be discussed briefly.

Algol 68 is defined in three stages:

a) The strict language.
b) The extended language.
c) The representation language.

The strict language is defined by syntax and semantic rules.

The syntax contains production rules for notions. A terminal production of a notion is a production which contains on its right side only symbols.

Example:

Conformity relator: conforms to symbol;

conforms to and becomes symbol.

The ";" means the logical "or".

The language is defined in such a way, that not all the productions of the strict language are given. There exists a method for converting the productions of the strict language using a meta-grammar, the productions of which are all given.

Example 1:

MODE: MOOD; UNITED;

is a rule of the meta-grammar.

Example 2:

FIGURE: zero; one; two; three; four; five; six; seven; eight; nine.

is a terminal production of a metanotion (FIGURE). The metanotions are distinguished from the notions by the fact that they are written in capital
letters. The production is terminal because on its right side appear only notions (in small letters). A production of the strict language in which the metanotion FIGURE appears is in reality a scheme generating ten productions, which can be obtained by replacing FIGURE with the possible terminal derivations. The number of notions of the strict language is infinite, because there are metanotions with an infinite number of terminal productions.

A program in the strict language conforms to the rules of the strict language.

A program is considered to be in extended language when it makes use of extensions. An example of an extension is the comment which in Algol 68 can appear almost everywhere.

Example:

`c this is a comment c`

The representation language represents the extended language.

Example:

The language symbols real and begin are represented for example by real, begin.

Facilities of the Language

Algol 68 is a universal programming language designed for different kinds of applications.

Its design philosophy is completely different from that of PL/I:

In PL/I almost everything goes while in Algol 68 everything has to be stated explicitly.
Example:

I=X; where I is integer and X real is allowed in PL/I but i:=x; is not allowed in Algol 68 and in order to perform this operation the operator entier has to be used.

The Mode Concept

Algol 68 generalizes the Algol 60 notion of type by introducing the concept of mode. Every kind of data is of a certain mode.

There are five primitive modes in the language. From these primitive modes other modes can be created using special symbols.

Once a new mode has been created, it is possible to declare variables of that mode.

Example:

mode string = [1:0 flex] char;

This is a mode declaration. The name of the created mode is string. Its meaning is that it stands for an unidimensional array of characters. The upper bound of the array is flexible and this declaration makes it temporarily equal to zero.

Note that char is a primitive mode. In order to declare a variable of the new mode the following declaration can be used:

string B;

This declaration creates a variable B which assumes values of mode string.
In Algol 68 there is an aggregate of data called structure. This aggregate is created from existing modes using the `struct` symbol.

Examples:

```plaintext
mode A = struct (int B, C, real D, ref A link);
mode E = struct (A L, bool x);
```

The mode A is a structure composed of two integers which can be selected by the field selectors B and C respectively, an element of mode real with the field selector D and a field of mode ref A having the field selector link.

The second declaration declares a new mode E, which is also a structure composed of two fields; one having the mode A with the field selector L, and the other a boolean with the field selector X.

Another available possibility in Algol 68 is to declare a mode, which is a union of modes, this means that a variable of this mode can be at any given moment from one of the modes that construct the union.

Example:

```plaintext
mode prim = union (int, real, bool, char);
```

The unions have the property of being commutative and associative. This means that the following modes are equivalent to the previous one:

```plaintext
union (real, bool, char, int) and
union (real, union (char, bool, int)).
```

The possibility of creating a mode, which is a union of existing modes, allows
us to write procedures, which can operate on arguments of different modes, by declaring the parameter of the procedure as a union of the desired modes.

Let us return to the symbol ref which was used in the example on structures. In Algol 68, variables as such do not exist. There are external objects and internal objects. Between these objects exist several relations. An external object can be an identifier used by the programmer.

Example:
real x;
In this example x is an external object.

Between the internal objects existing in the computer during the elaboration of a program, there might exist a relation of reference. This means that one internal object refers to another. An internal object which refers to a value of mode real is of mode ref real.

An object of mode ref real is called in other languages a variable. An object of mode ref ref real is called in other languages a pointer. The level of possible indirectness is not limited by the language. It is easy to do list processing with this tool.

Another relation of internal objects is that of "being of the same mode". This relation has to be tested for example in a procedure designed for working on objects of different modes.
Example:

\texttt{int::i}

The above mentioned example shows the use of the conformity operator. The result of the test is \texttt{true} if \texttt{i} is of mode integer and \texttt{false} otherwise.

An internal object that refers to another internal object is called a \texttt{name}.

Sometimes one is interested to know whether two names are the same. This is done by using the operators \texttt{::=} (is) and \texttt{:#} (is not).

Example:

\texttt{(ref real : XX) ::= (ref real : yy)};

The two points, which appear between the mode and the identifier are called cast.

The cast is used in order to force conversions (coercions) from one mode to another. It is said that \texttt{xx} and \texttt{yy} are in strong positions, which means that all eight available kinds of coercions in the language can be applied.

\textbf{Algol 68} introduces two new concepts, which did not exist in Algol 60.

They are the format and the routine.

The format is used by the I/O procedures. The routine is a piece of program which is considered to be a value like say 3.14.

An external object like a procedure name or an operator name can "possess" a routine. The relation of possession exists between external and internal objects. One is allowed in Algol 68 to declare a procedure variable, which can change its body during the elaboration of the program.

The change is done by assigning to the variable a new routine.
Algol 68 allows the declaration of new operators which are transformed to key words for the particular run.

**Example:**
Suppose that one wishes to declare an operator "ip", which works on integer numbers and which gives the value **true** if the number can be divided by seven and **false** otherwise.

```
    op ip = (int a) bool : (a mod 7=0 | true | false);
```

In the program I can write:

```
    bool b:=ip C;
```

The action of this statement is the creation of a boolean variable of mode **ref bool** which is initialized to **true** if C can be divided by seven and **false** otherwise. The above declared operator is monadic (unary). The language allows also the declaration of dyadic operators and even the declaration of the same indicant (representation) of an operator which works on different modes.

**Example:**
```
    op sub = (int a, b) int : (a+b) * 2;
    op sub = (real a, b) real : (a+b) * 3;
```

If in the program an expression like `c sub d` appears, then if both C and D are integers the first operation will be performed, and if they are both real the second operator will be applied.

In the case that C and D are of different modes and if no adequate operator declaration exists the use of `sub` will be incorrect.
The evaluation of expressions is done using the priority of the operators. All monadic operators have the highest priority, but a possibility exists for changing the priority of existing operators or declaring priorities for new ones.

Example:

```
Priority sub = 2;
```

Another important concept in Algol 68 is the disappearance of the distinction between expressions and instructions. A unit of program called a *closed clause* has as value the value of the last statement executed. This gives us the possibility of combining closed clauses in expressions.

Example:

```
a := a + (real x ; x := a ↑ 2 ; x );
```

The closed clause in the example is located between the parenthesis. The value of it is the value of x.

In Algol 60 the termination of a procedure was reached at the *end* or when terminating the evaluation of the statement if the procedure had only one. If one wanted to skip several statements he gave a label to the *end* and performed an unconditional jump to it. In order to achieve the same result in Algol 68 we use the exit symbol. The effect of it is the same as the RETURN of FORTRAN and PL/I.

Another step forward is achieved in the area of multiple values of the same mode, known commonly as arrays.

In Algol 60, and Fortran we have access only to an element of the array. In PL/I there is more flexibility and we can deal with rows and columns of a
matrix (and arrays of more dimensions). In Algol 68 through the use of "slices" we have even more freedom.

**Example:**
Suppose the following declaration exists:

\[
\begin{bmatrix}
1:10, 1:10
\end{bmatrix}
\text{real } A;
\]

which is equivalent to the Algol 60 declaration:

\[
\text{real array } A \begin{bmatrix}
1:10, 1:10
\end{bmatrix};
\]

I can address an element \texttt{Ex A[2,4]} and use slices in order to address parts of the array \texttt{A}.

**Examples:**

a) \texttt{A[2:4,1:3]}
b) \texttt{A[2,1:5]}
c) \texttt{A[3:,1:5]}
d) \texttt{A[,,4]}

In the first example the slice consists from the elements: \texttt{A[2,1]}, \texttt{A[2,2]}, \texttt{A[2,3]}, \texttt{A[3,1]}, \texttt{A[3,2]}, \texttt{A[3,3]}, \texttt{A[4,1]}, \texttt{A[4,2]}, \texttt{A[4,3]}.

In the second example we address the first five elements of the second row namely: \texttt{A[2,1]}, \texttt{A[2,2]}, \texttt{A[2,3]}, \texttt{A[2,4]}, \texttt{A[2,5]}.

In the third example the slice is composed from rows 3 to 10 and in each row we take the first five elements.

In the fourth example we address the fourth column.

**Input - Output**
Contrarily to Algol 60 which did not include specific I/O, Algol 68 defines the topic extensively.

A program communicates with its environment using "channels". A channel can be either one device or a group of devices ex: tapes, cards and discs. There are two kinds of transput (I/O). One in character format where the external
representation is readable like for printing and the second called the binary format where the representation is not readable.

Special procedures are designed for testing the state of the channels.

Example:
"get possible" - it is possible to issue read commands for the channel.
"put Possible" - it is possible to issue write commands for the channel.
"reset possible" - repositioning of files is possible on this channel.
"set possible" - direct access is possible on the channel.
"idf possible" - labels can be attached to files for identifying purposes.
"reidf possible" - it is possible to modify the labels.

Three standard files are at the disposal of the user.

a) A file for reading. Its name is "standin" and this is essentially the SYSIN of PL/I.

b) A file for writing. Its name is "standout" and this is essentially the SYSPRINT of PL/I.

c) A file for intermediate work. Its name is "standback". There is no similar concept in PL/I.

Input - Output on the standard files is effected using the procedures "read" and "print".

On other files created by the programmer the I/O is effected using the procedures "put" and "get". Opening and closing procedures are supplied which perform the action of OPEN and CLOSE of PL/I. Like in PL/I, the user can provide procedures for dealing with errors in I/O. Transput can be done either in free or variable formats.
Example:

print("ABC");
This order will print the string "ABC" on the standard output file "Standout".
This order is equivalent to the PL/I statement:

PUT LIST ('ABC');

Several items can be printed with one order by enclosing them in parenthesis.

Example:

Print((y,i+5, a ∨ b ∧ c, "abc"));

This order prints four items on "standout". Note that the items are enclosed in two pairs of parenthesis. The outer pair is for the call of the procedure "print" and the inner pair is needed because the procedure 'print' has only one parameter. The argument in this case is called a row display.

Special procedures give you the possibility to skip to a new page, skip to a new line, and advance in the same line and going back in the same line. These procedures can be called during the use of printing orders.

Special procedures allow you to make I/O with variable formats. The formats can be specified at execution time.

Conclusion

This paper did not attempt to discuss in detail all the features of Algol 68. Especially very little was said about the transput facilities of the language. It is hoped that the reader will get interested in the language and look for more information in the documents cited in the bibliography. It is also hoped that this short informal introduction will help in understanding the formal definition of the language.
BIBLIOGRAPHY

Report on the Algorithmic language Algol 68

Informal Introduction to Algol 68.
4 draft August 1969.