Information Processing in Japan Vol. 11, 1971

# On the Formal Grammatical Structure of ALGOL 60

MAKOTO ARISAWA*

*Abstract*

Naur described the syntax of ALGOL 60 using BNF notation. In this paper, the formal grammatical structure of this description is investigated. This description seems not to be convenient for computer's use, because there are a lot of redundant variables and rules which are intended to be man's use. All the variables can be classified into three groups, and we can represent the syntax structure using regular expressions in each group.

## 1. *Introduction*

Recently there published a lot of reports on syntax-directed compilers, and BNF notation has been occupying an important role in most of them. BNF was first used to describe the syntax of ALGOL 60, and since then it has influenced on almost all the meta languages. Moreover BNF just corresponds to CFG in formal language theory, there is a hope that we can use theorems in formal language theory to programming language processing.

But the language class defined by CFG is not necessarily equal to the class of programming languages now used, since there are both aspects that CFG covers wider ranges in some cases but narrower in another cases. The author does not think BNF is powerful tool for practical syntax-directed compilers.

In the present paper, we investigate the formal grammatical structure of ALGOL 60, and make it clear what position the BNF notation occupies, and what kind of meta language is practical.

## 2. *Terminology*

Most of the words and phrases used in this paper follow the definitions in [3].

The "recursive variable" is a non-terminal symbol in BNF notation, which appears on both sides of a rule.

The "essential variable" is a non-terminal symbol in BNF notation, which remains after erasing non-terminal symbols by replacing a non-terminal on right side of rules with a whole right side of the rule in which the non-terminal is defined. Notice that a recursive variable is an essential variable, but the converse is not true.

This paper first appeared in Japanese in Joho Shori (the Journal of the Information Processing Society of Japan), Vol. 11, No. 9 (1971), pp. 509-516.

* Electrotechnical Laboratory

Table 1. Characteristics of Variables

| variable | ① | ② | ③ | ④ | ⑤ | | |
|---|---|---|---|---|---|---|---|
| ⟨actual parameter⟩ | 1 | 2 | 5 | | | 12 | |
| ⟨actual parameter list⟩ | 2 | 2 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle ba$ | | 13 | |
| ⟨actual parameter part⟩ | 2 | 2 | 2 | | | 14 | |
| ⟨adding operator⟩ | 1 | 2 | 2 | | 1 | | |
| ⟨ang sequence of basic symbols not containing 'or'⟩ | 1 | 1 | / | | 0 | | |
| ⟨arithmetic expression⟩ | 8 | 12 | 2 | $\langle A\rangle ::= a \mid bac\langle A\rangle$ | | ⑩ | |
| ⟨arithmetic operator⟩ | 1 | 1 | 6 | | 1 | | |
| ⟨array declaration⟩ | 1 | 1 | 2 | | | 16 | |
| ⟨array identifier⟩ | 3 | 4 | 1 | | 3 | | |
| ⟨array list⟩ | 2 | 3 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle ba$ | | 15 | |
| ⟨array segment⟩ | 2 | 3 | 2 | $\langle A\rangle ::= ab \mid ac\langle A\rangle$ | | 14 | |
| ⟨assignment statement⟩ | 1 | 1 | 2 | | | 17 | |
| ⟨basic statement⟩ | 2 | 2 | 2 | $\langle A\rangle ::= a \mid b\langle A\rangle$ | | 19 | |
| ⟨basic symbol⟩ | 0 | 0 | 4 | | 4 | | |
| ⟨block⟩ | 3 | 3 | 2 | $\langle A\rangle ::= a \mid b\langle A\rangle$ | | | 57 |
| ⟨block head⟩ | 2 | 2 | 2 | $\langle A\rangle ::= ab \mid \langle A\rangle cb$ | | | 55 |
| ⟨Boolean expression⟩ | 6 | 6 | 2 | $\langle A\rangle ::= a \mid bac\langle A\rangle$ | | ⑩ | |
| ⟨Boolean factor⟩ | 2 | 3 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle ba$ | | 23 | |
| ⟨Boolean primary⟩ | 1 | 2 | 5 | | | 21 | |
| ⟨Boolean secondary⟩ | 1 | 2 | 2 | | | 22 | |
| ⟨Boolean term⟩ | 2 | 3 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle ba$ | | 24 | |
| ⟨bound pair⟩ | 1 | 2 | 1 | | | 12 | |
| ⟨bound pair list⟩ | 2 | 2 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle ba$ | | 13 | |
| ⟨bracket⟩ | 1 | 1 | 8 | | 1 | | |
| ⟨code⟩ | 1 | 1 | / | | 0 | | |
| ⟨compound statement⟩ | 3 | 3 | 2 | $\langle A\rangle ::= a \mid b\langle A\rangle$ | | | 53 |
| ⟨compound tail⟩ | 3 | 3 | 2 | $\langle A\rangle ::= ab \mid ac\langle A\rangle$ | | | 52 |
| ⟨conditional statement⟩ | 2 | 2 | 4 | $\langle A\rangle ::= a \mid ab \mid c \mid d\langle A\rangle$ | | | ㊿ |
| ⟨decimal fraction⟩ | 1 | 2 | 1 | | 3 | | |
| ⟨decimal number⟩ | 1 | 2 | 3 | | 4 | | |
| ⟨declaration⟩ | 1 | 2 | 4 | | | | 54 |
| ⟨declarator⟩ | 1 | 1 | 7 | | 1 | | |
| ⟨delimiter⟩ | 1 | 1 | 5 | | 3 | | |
| ⟨designational expression⟩ | 5 | 6 | 2 | $\langle A\rangle ::= a \mid bac\langle A\rangle$ | | ⑩ | |
| ⟨digit⟩ | 3 | 4 | 10 | | 1 | | |
| ⟨dummy statement⟩ | 1 | 1 | 1 | | 2 | | |
| ⟨empty⟩ | 6 | 6 | 1 | | 1 | | |
| ⟨exponent part⟩ | 1 | 2 | 1 | | 4 | | |
| ⟨expression⟩ | 1 | 1 | 3 | | | 11 | |
| ⟨factor⟩ | 2 | 3 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle ba$ | | 17 | |
| ⟨for clause⟩ | 1 | 1 | 1 | | | 15 | |
| ⟨for list⟩ | 2 | 2 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle ba$ | | 12 | |
| ⟨for list element⟩ | 1 | 2 | 3 | | | 11 | |
| ⟨formal parameter⟩ | 1 | 2 | 1 | | 3 | | |
| ⟨formal parameter list⟩ | 2 | 2 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle ba$ | 4 | | |
| ⟨formal parameter part⟩ | 1 | 1 | 2 | | 5 | | |
| ⟨for statement⟩ | 3 | 3 | 2 | $\langle A\rangle ::= a \mid b\langle A\rangle$ | | | ㊿ |
| ⟨function designator⟩ | 2 | 2 | 1 | | | 15 | |
| ⟨goto statement⟩ | 1 | 1 | 1 | | | 11 | |
| ⟨identifier⟩ | 8 | 10 | 3 | $\langle A\rangle ::= a \mid \langle A\rangle a \mid \langle A\rangle b$ | 2 | | |
| ⟨identifier list⟩ | 3 | 4 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle ba$ | 3 | | |
| ⟨if clause⟩ | 5 | 5 | 1 | | | 11 | |
| ⟨if statement⟩ | 1 | 2 | 1 | | | | 51 |
| ⟨implication⟩ | 2 | 3 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle ba$ | | 25 | |
| ⟨integer⟩ | 1 | 1 | 3 | | 3 | | |

| | ① | ② | ③ | ④ | ⑤ | | |
|---|---|---|---|---|---|---|---|
| ⟨label⟩ | 6 | 6 | 2 | | 3 | | |
| ⟨left part⟩ | 1 | 2 | 2 | | | 15 | |
| ⟨left part list⟩ | 2 | 3 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle a$ | | 16 | |
| ⟨letter⟩ | 3 | 5 | 52 | | 1 | | |
| ⟨letter string⟩ | 2 | 2 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle a$ | 2 | | |
| ⟨localor or own type⟩ | 2 | 2 | 2 | | 2 | | |
| ⟨logical operator⟩ | 1 | 1 | 5 | | 1 | | |
| ⟨logical value⟩ | 2 | 2 | 2 | | 1 | | |
| ⟨lower bound⟩ | 1 | 1 | 1 | | | 11 | |
| ⟨multiplying operator⟩ | 1 | 1 | 3 | | 1 | | |
| ⟨number⟩ | 0 | 0 | 3 | | 6 | | |
| ⟨open string⟩ | 2 | 4 | 3 | $\langle A\rangle ::= a \mid b\langle A\rangle c \mid \langle A\rangle\langle A\rangle$ | | ⑩ | |
| ⟨operator⟩ | 1 | 1 | 4 | | 2 | | |
| ⟨parameter delimiter⟩ | 2 | 2 | 2 | | 3 | | |
| ⟨primary⟩ | 1 | 2 | 4 | | | 16 | |
| ⟨procedure body⟩ | 1 | 2 | 2 | | | | 52 |
| ⟨procedure declaration⟩ | 1 | 1 | 2 | | | | 53 |
| ⟨procedure heading⟩ | 1 | 2 | 1 | | 6 | | |
| ⟨procedure identifier⟩ | 5 | 5 | 1 | | 3 | | |
| ⟨procedure statement⟩ | 1 | 1 | 1 | | | 15 | |
| ⟨program⟩ | 0 | 0 | 2 | | | | 58 |
| ⟨proper string⟩ | 1 | 1 | 2 | | 2 | | |
| ⟨relation⟩ | 1 | 1 | 1 | | | 20 | |
| ⟨relational operator⟩ | 2 | 2 | 6 | | 1 | | |
| ⟨separator⟩ | 1 | 1 | 11 | | 1 | | |
| ⟨sequential operator⟩ | 1 | 1 | 6 | | 1 | | |
| ⟨simple arithmetic expression⟩ | 3 | 5 | 3 | $\langle A\rangle ::= a \mid ba \mid \langle A\rangle ba$ | | 19 | |
| ⟨simple Boolean⟩ | 2 | 3 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle ba$ | | 26 | |
| ⟨simple designational expression⟩ | 1 | 2 | 3 | | | 13 | |
| ⟨simple variable⟩ | 2 | 3 | 1 | | 4 | | |
| ⟨specification part⟩ | 2 | 2 | 3 | $\langle A\rangle ::= a \mid b \mid \langle A\rangle b$ | 4 | | |
| ⟨specificator⟩ | 1 | 1 | 3 | | 1 | | |
| ⟨specifier⟩ | 1 | 2 | 8 | | 2 | | |
| ⟨statement⟩ | 4 | 5 | 3 | | | | 51 |
| ⟨string⟩ | 1 | 1 | 1 | | | 11 | |
| ⟨subscripted variable⟩ | 1 | 1 | 1 | | | 13 | |
| ⟨subscript expression⟩ | 2 | 3 | 1 | | | 11 | |
| ⟨subscript list⟩ | 2 | 2 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle ba$ | | 12 | |
| ⟨switch declaration⟩ | 1 | 1 | 1 | | | 12 | |
| ⟨switch designator⟩ | 1 | 1 | 1 | | | 12 | |
| ⟨switch identifier⟩ | 3 | 3 | 1 | | 3 | | |
| ⟨switch list⟩ | 2 | 2 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle ba$ | | 11 | |
| ⟨term⟩ | 2 | 4 | 1 | $\langle A\rangle ::= a \mid \langle A\rangle ba$ | | 18 | |
| ⟨type⟩ | 3 | 6 | 3 | | 1 | | |
| ⟨type declaration⟩ | 1 | 1 | 1 | | 6 | | |
| ⟨type list⟩ | 2 | 2 | 2 | $\langle A\rangle ::= a \mid ab\langle A\rangle$ | 5 | | |
| ⟨unconditional statement⟩ | 2 | 2 | 3 | | | | ㊿ |
| ⟨unlabelled basic statement⟩ | 1 | 1 | 4 | | | 18 | |
| ⟨unlabelled block⟩ | 1 | 1 | 1 | | | | 56 |
| ⟨unlabelled compound⟩ | 1 | 1 | 1 | | | | 53 |
| ⟨unsigned integer⟩ | 5 | 8 | 2 | $\langle A\rangle ::= a \mid \langle A\rangle a$ | 2 | | |
| ⟨unsigned number⟩ | 2 | 4 | 3 | | 5 | | |
| ⟨upper bound⟩ | 1 | 1 | 1 | | | 11 | |
| ⟨value part⟩ | 1 | 1 | 2 | | 4 | | |
| ⟨variable⟩ | 4 | 4 | 2 | | | 14 | |
| ⟨variable identifier⟩ | 1 | 1 | 1 | | 3 | | |

① number of rules in which the variable is refered
② number of reference of the variable
③ number of selection of right side of the rule
④ type of recursive rule
⑤ level number

3. *Characteristics of the rules in ALGOL 60*

The BNF description of ALGOL 60 is composed of 109 rules with 111 variables. In the Naur's report [2], seven rules appear in two different places.

First we list up how many times each variable is refered in the right side of the rules, and in how many rules it is refered. Frequently refered variables contribute much to the grammatical structure.

Next we check the number of selections of the right side of the rule whose left side contains the variable. If this value is large then the variable represents a wide concept in the grammar.

These numbers are shown in Table 1. In the table, we see that a lot of variables are rather redundant, and do not much contribute to the structural composition. Their main role may be to help people's understanding.

4. *Recursive rule types*

In the BNF representation of ALGOL 60, in addition to the starting variable there are 33 essential variables, all of which are recursive variables. We check those rules in which recursive variable appears on the leftside, and classify them. One rule for ⟨open string⟩ falls into a special group as

⟨open string⟩ ::=⟨proper string⟩|'⟨open string⟩'|
⟨open string⟩ ⟨open string⟩.

Other 32 rules are devided into three types as follows:

Type $A$ $\langle A\rangle ::= a \mid \langle A\rangle a$

Type $B$ $\langle A\rangle ::= a \mid b\langle A\rangle$

Type $C$ $\langle A\rangle ::= a \mid \langle A\rangle ba$

or $\langle A\rangle ::= a \mid ab\langle A\rangle$.

Type $A$ shows concatenation of the same terms. Type $B$ is the same as Type $A$ except that the last term is different from others. Type $C$ shows that a punctuation mark is used between the concatenated terms. Examples for each type are, ⟨unsigned integer⟩, ⟨basic statement⟩, and ⟨identifier list⟩ respectively. Most of the recursive variables are classified to Type $C$. Those which are not classified exactly into above groups are considered to be variations of above three types. Table 1 also contains this information.

Generally this type of rule is represented in a tree of Fig. 1. The same tree can be rewritten as Fig. 2. In the latter case, we may express the tree as

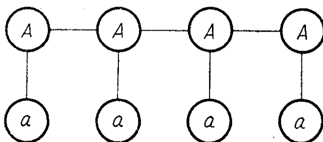


Fig. 1. Tree structure for the rule. $\langle A\rangle ::= a \mid a\langle A\rangle$.

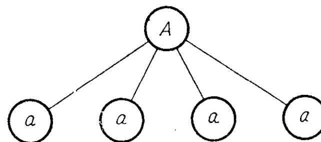


Fig. 2. The structure for the rule $\langle A\rangle = aa^*$.

⟨$A$⟩$=aa^*$, using regular expression. It is suggested to use regular expression instead of BNF. This can only be done when we see the rules locally, and it is not true that ALGOL 60 is described only by regular expression.

5. *Variable levels*

In this section, we try to describe rules in regular expression, starting from the variables near to the terminal symbols. If in a rule

⟨$A$⟩ ::=⟨$B$⟩|⟨$A$⟩ ⟨$B$⟩,

⟨$B$⟩ can be expressed by a regular expression $u$, then ⟨$A$⟩$=uu^*$.

We start from the terminal symbols, using above method, up to higher variables. Practically we assign a "level number" to each variable. Terminal symbols are assigned level number 0. The level number of a variable is calculated as maximum level number which appears on the right side of the rule whose left side is the variable, plus one. In case of the recursive variables, maximum is taken without considering the variable for which we are going to assign the level number. If we cannot assign the level number any more, we stop there.

Starting from the terminal symbols, we can assign 47 variables with 1 to 6. Seven recursive variables such as ⟨identifier⟩, ⟨type list⟩ are included in these variables. These are considered to be variables which can be expressed only by regular expression.

In the remaining variables are included variables concerning expressions and we give level number 10 to the three variables which appear on the right side of the rule,

⟨expression⟩ ::=⟨arithmetic expression⟩|
⟨Boolean expression⟩|
⟨designational expression⟩.

We also give the level number 10 to the variable ⟨open string⟩. We can continue the previous process until we assign level numbers from 11 to 26, for another 47 variables. Then we give the level number 50 to the three variables which appear on the right side of the rule,

⟨statement⟩ ::=⟨unconditional statement⟩|
⟨conditional statement⟩|⟨for statement⟩.

Repeating the same process, we can assign remaining 15 variables and terminate by assigning 58 to ⟨program⟩.

The variables are then divided into three groups, level number 1 to 6, 10 to 26, and 50 to 58. The first group is usually analyzed by a lexical analyzer in a compiler. The second group is expression part, and the third is statement part, both of them being analyzed by a syntax analyzer. We take an example from the second group. A simple expression $A*B+C$ has a syntax tree shown in

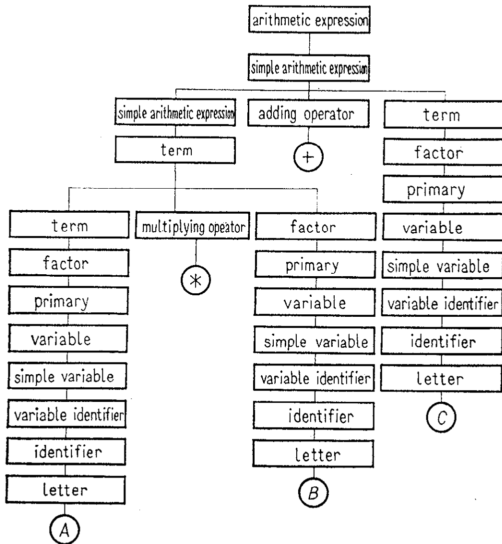


Fig. 3. Formal syntax tree for the expression $A*B+C$.

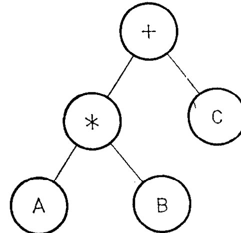


Fig. 4. Basic syntax tree for the expression $A*B+C$.

Fig. 3. The same tree can be expressed as Fig. 4. The structure of the expression has two points, nesting property and operator precedence. Fig. 3 seems rather complex because it represents the precedence relation perfectly. One advantageous feature of BNF notation is that it can clearly express the precedence. But its complexity makes it disadvantageous to use BNF notation directly to compose a syntax tree. To process an expression, it is more efficient

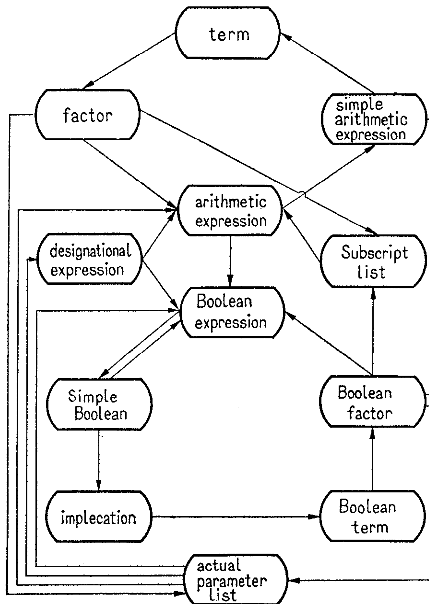


Fig. 5. Relation between some essential variables.

to use such method as operator precedence method, and not syntax directed method. In Fig. 5, the relation between essential variables of the second group is shown. There exists an arrow from variable ⟨A⟩ to ⟨B⟩ when ⟨B⟩ is necessary to define ⟨A⟩.

In the third group, the relation is also complex. If we consider the variable ⟨statement⟩ as a terminal, we can express this group by regular expression, as follows:

⟨s.⟩ ::=L*FC⟨s.⟩|L*BS|L*⟨compound s.⟩|
L*⟨block⟩|L*IC BS|
L*IC⟨compound s.⟩|L*IC⟨block⟩|
L*IC BS **else** ⟨s.⟩|
L*IC⟨compound s.⟩**else**⟨s.⟩|
L*IC⟨block⟩**else**⟨s.⟩|L*ICL*FC⟨s.⟩
⟨block⟩ ::=L***begin**⟨d.⟩[;⟨d.⟩]*[⟨s.⟩;]*
⟨s.⟩**end**
⟨compound s.⟩ ::=L***begin**[⟨s.⟩;]*⟨s.⟩**end**
⟨d.⟩ ::=⟨type d.⟩|⟨array d.⟩|⟨switch d.⟩|
**procedure** PH⟨s.⟩|**procedure** PH
⟨code⟩|⟨type⟩**procedure** PH⟨s.⟩|
⟨type⟩**procedure** PH⟨code⟩
⟨program⟩ ::=⟨block⟩|⟨compound s.⟩

Here, L, IC, FC, BS, PH, ⟨s.⟩, ⟨d.⟩ are abbreviations of, ⟨label⟩, ⟨if clause⟩, ⟨for clause⟩, ⟨basic statement⟩, ⟨procedure heading⟩, ⟨statement⟩, and ⟨declaration⟩ respectively.

## 6. *Conclusion*

As a result of the above sections, we see two points. One is that BNF representation of ALGOL 60 has many redundant variables. The other is that variables are divided into three groups, one for lexical analysis part, one for expression part, and the other for statement part. Within each group, we can express the relation of the variables using regular expression.

Author and his colleagues designed ESDL (ETL's System Description Language) system, and experimentally implemented its compiler [8]. In the compiler, analyzing part were divided just along the above three groups.

There have been some trials to use regular expression instead of BNF in [7], and this approach seems to the author very promising.

*Acknowledgement*

The author thanks Dr. K. Noda, Dr. H. Nishino, Dr. H. Aiso, K. Fuchi, Dr. K. Torii, N. Saito, K. Furukawa, Y. Sugito, M. Miyakawa, M. Sato and all the

members of our group for their suggestions and discussions.

*References*

[1] Moriguchi, S.: Introduction to ALGOL. JUSE Pub (1962).
[2] Naur, P.: Revised report on ALGOL 60 (1962).
[3] Ginsburg, S.: The mathematical theory of CFL. Mc-Graw Hill (1966).
[4] Hennie, F. C.: Finite state models for logical machines. John Wiley (1968).
[5] Knuth, D. E.: The art of computer programming, Vol. 1. Addison Wesley (1968).
[6] Hopgood, F.: Compiling techniques. Elsevier (1968).
[7] Feldman, J., et al.: Translator writing systems. CACM-11 (1968).
[8] Arisawa, M., et al.: On ESDL compiler, and other papers. Bul. ETL-34 (1970).