1. Syntax

1.1 Meta-Language for Syntax Description

To describe the syntax of ALGOL N we use the following meta-language, which is a simple modification of the Backus notation.

1.1.1 Meta-symbols: :=, [, ], { }, |, / , ....

1.1.2 Meta-constants: begin, ;, ( , a, etc.

Those are basic symbols of ALGOL N.

1.1.3 Meta-variables: \langle expression\rangle, \langle variable\rangle, \langle procedure call\rangle etc.

Those are used to represent the form of syntactical elements.

1.1.4 Meta-expressions:

A meta-expression represents the forms of figures which are finite sequences of basic symbols, and are defined recursively as follows:

1) Let \( a \) stand for a meta-constant. Then

\[ a \]

is a meta-expression.

A figure \( \phi \) is of the form \( a \), if and only if \( \phi \) consists of just one basic symbol \( a \).

2) Let \( a \) stand for a meta-variable. Then

\[ a \]

is a meta-expression.

A figure \( \phi \) is of the form \( a \), if and only if \( \phi \) is of the form meta-variable \( a \).

3) Let \( a \) stand for a meta-expression. Then

\[ (a) \]

is a meta-expression, and is used to express the precedence of connections.

A figure \( \phi \) is of the form \( (a) \), if and only if \( \phi \) is of the form \( a \).

Syntax-1
4) Let $\alpha$ stand for a meta-expression. Then 

$$[\alpha]$$

is a meta-expression.

A figure $\psi$ is of the form $[\alpha]$, if and only if $\psi$ is the empty figure or is of the form $\alpha$.

5) Let $\alpha$, $\beta$ stand for meta-expressions. Then 

$$\alpha \beta$$

is a meta-expression.

A figure $\psi$ is of the form $\alpha \beta$, if and only if $\psi$ is the concatenation of a figure of the form $\alpha$ and a figure of the form $\beta$.

6) Let $\alpha$, $\beta$ stand for meta-expressions. Then 

$$\alpha \| \beta$$

is a meta-expression.

A figure $\psi$ is of the form $\alpha \| \beta$, if and only if either $\psi$ is of the form $\alpha$ or $\psi$ is of the form $\beta$.

7) Let $\alpha$ stand for a meta-expression. Then 

$$\alpha \ldots$$

is a meta-expression.

A figure $\psi$ is of the form $\alpha \ldots$, if and only if either $\psi$ is of the form $\alpha$ or $\psi$ is the concatenation of a figure of the form $\alpha \ldots$ and a figure of the form $\alpha$.

8) Let $\alpha$, $\beta$ stand for meta-expressions. Then 

$$\alpha \{\beta\} \ldots$$

is a meta-expression.

A figure $\psi$ is of the form $\alpha \{\beta\} \ldots$, if and only if either $\psi$ is of the form $\alpha$ or $\psi$ is the concatenation of a figure of the form $\alpha \{\beta\} \ldots$, a figure of the form $\beta$, and a figure of the form $\alpha$.

9) Let $\alpha$, $\beta$ stand for meta-expressions. Then

Syntax-2
A figure $\phi$ is of the form $a/\beta$, if and only if $\phi$ is of the form $a\{\beta\}$.

The ranking of the priorities of connections is as follows:

first: $a\ldots$, $a/\beta\ldots$

second: $a\beta$

third: $a\{\beta\}, a/\beta$

1.1.5 Meta-statement:

A meta-statement is used to define a meta-variable. Let $a$ stand for a meta-variable, and $\beta$ stand for a meta-expression. Then

$a \equiv \beta$

is a meta-statement. This meta-statement represents the sentence:

"A figure $\phi$ is of the form of $a$, if and only if $\phi$ is of the form $\beta$.

In the following, we shall say simply

"$\phi$ is $a$" in stead of "$\phi$ is of the form $a$".
1.2 Standard Language of ALGOL N

<expression>  == <secondary>  |
              <form call>

<secondary>  == <primary>  |
              <array element>  |
              <structure element>  |
              <procedure call>

<primary>  == <variable>  |
            <go to statement>  |
            <dummy statement>  |
            <code call>  |
            <closed expression>  |
            <block>  |
            <notation>

<notation>  == <effect notation>  |
            <real notation>  |
            <bits notation>  |
            <string notation>  |
            <reference notation>  |
            <array notation>  |
            <structure notation>  |
            <procedure notation>

<declaration>  == <variable declaration>  |
                 <form declaration>  |
                 <mark declaration>

Syntax—4
<go to statement>  == go to <label>
<dummy statement>  == dummy
<code call>        == code ( [ <selector> <expression> ] , ... )
                   
                   <primary typifier> by ( <code body> )
<closed expression> == ( <expression> )
<block>            == begin [ <declaration> ; ] ... 
                   
                   ( [ <label> : ] ... <expression> ) ; ... end
<array element>    == <secondary> [ <expression> ]
<structure element>== <secondary> [ <selector> ]
<procedure call>   == <secondary> ( [ <expression> , ... ] )
<form call>       == [ <expression> ] <mark> [ <expression> ] <mark> ; ... 
<effect notation>  == effect
<real notation>    == real <real modifier> <real donor>
<real modifier>    == [ L [ <expression> ] : [ <expression> ] :
                   
                   [ <expression> ] ] ] :
                   
                   [ precision [ <expression> ] ] ]
<real donor>       == [ <number> ]
<number>           == <integer donor> | <fraction donor>
<integer donor>    == <digit> ...
<fraction donor>   == [ <digit> ... ]( . <digit> ... / (10^+10^-) <digit> ... )
<bits notation>  = bits <bits modifier> <bits donor>
<bricks modifier>  = [ (exact | varying) [ <expression> ] ]
<bricks donor>  = ['<bits> ']
<bricks>  = (0 | 1) ...
<string notation>  = string <string modifier> <string donor>
<string modifier>  = [ (exact | varying) [ <expression> ] ]
<string donor>  = ['<string> ']
<string>  = "[<non"'>]<string>..."
<reference notation>  = reference <reference donor>
<reference donor>  = ['nil ']
<array notation>  = array <array modifier> <primary> |
<array modifier>  = [ (expression) : <expression> ]
<structure notation>  = structure [ [(selector) <expression>],..., ]
<procedure notation>  = procedure [ [typifier],... ]
<procedure donor>  = [ by [[variable],..., ] <expression> ]
<variable declaration> = let <variable> be <expression>
<form declaration> = let <form> represent <expression>
<form>  = [('<message> )]<mark>['(message)',] ']<mark>...
<mark declaration> = let <mark> operate <left priority> <right priority>

<left priority> = [before ([<mark>, ...] all) left]
<right priority> = [after ([<mark>, ...] all) right]
<typifier> = <expression>
<primary typifier> = <primary>
<variable> = <identifier>
<label> = <identifier> ↓
<identifier> = <letter> [<letter> | <digit>]...
<letter> = a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
<digit> = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<basic symbol> = <non'>' | ' ' |
<non'>' = <delimiter> | <letter> | <digit> | . | _ | _ | _ | _ | _ | _ | _ | _ |
<delimiter> = <standard symbol> | <extension symbol> | <mark>
<standard symbol> = begin | end | ( ) [ ] { } | before | left | after | right | effect | real | bits | string | reference | array | structure | procedure | precision | exact | varying | nil | code | by | let | be | represent | operate | all | go to | dummy | ; | :: | , | -

Syntax—7
<extension symbol> = integer | none | comment | silent | 2 array | etc.

<mark> = := | if | then | else | for | do | from | step | until
while | case | of | + | - | \ | \ | / | % | \ | / | % | ≤ | ≥ | < | ≤ | = | ≥ 
> | ≠ | complex | copy | enproc | mode | length | bd | succ
the | ref | has type | as type | etc.

<code body> is not specified.
1.3 Extensions

1.3.1 When $E_1$ is empty or an expression for $i = 1, 2$, "real $[E_1 : E_2]$" may be replaced by "integer $[E_1 : E_2]$".

1.3.2 "integer $[ : ]$" may be replaced by "integer".

1.3.3 When $J$ is a <number>, "real $J$" may be replaced by "J".

1.3.4 "precision" may be omitted.

1.3.5 When $J$ is a <bits>, "bits $J$" may be replaced by "J".

1.3.6 When $J$ is a <string>, "string $J$" may be replaced by "J".

1.3.7 "exact" may be omitted.

1.3.8 "reference nil" may be replaced by "nil".

1.3.9 "array $[$" may be replaced by "[".

1.3.10 "structure $[$" may be replaced by "{|".

1.3.11 When $E_1$ and $E_2$ are expression's, "$[E_1 : E_2]$ array $[$" may be replaced by "array $[E_1 : E_2]$ array $[$".

1.3.12 When $E_1$ and $E_2$ are expression's, "$[E_1 : E_2]$ array array $[$" may be replaced by "array $[E_1 : E_2]$ array $[$".

1.3.13 "$ ][$" may be replaced by "[,]".

1.3.14 "array array" may be replaced by "$array$".

1.3.15 "$ ) effect$" may be replaced by "$ )$".

1.3.16 "$ ) effect$" may be replaced by "$ )$".

1.3.17 When $V_1$, ..., $V_n$ are <variable>'s and $E$ is an expression,

"let $V_1$ be $E$ ;
let $V_2$ be $E$ ;
....
let $V_n$ be $E$ ;"

may be replaced by

"let $V_1$, $V_2$, ..., $V_n$ be $E$ ;".

1.3.18 Let $V$ be a <variable> or a sequence of <variable>'s separated by commas, and $E$ be an <expression>. If the right-most symbol of $E$ is
effect, real, integer, bits, string, reference, nil, end, l, l, l, l or
then "let V be E ; "may be replaced by"EV ; ."

1.3.19 Let $T_k$ be a <typifier> for $i=1,2,\ldots,n$;
\[
P_i = \begin{cases} 
\text{a sequence of } \langle \text{mark} \rangle \text{'s for } i=1,2,\ldots,n-1; \\
\text{empty or a sequence of } \langle \text{mark} \rangle \text{'s for } i=0, n. 
\end{cases}
\]
"let $P_o(T_1)P_1(T_2)P_2\ldots P_{n-1}(T_n)P_n$ represent" may be replaced by
"let $P_o( )P_1( )P_2\ldots P_{n-1}( )P_n$ represent".

1.3.20 When G is a sequence of mark 's and parentheses, and $E_1,\ldots,E_n$
are <expression>'s,
\[
\text{"let G represent } E_1; \\
\text{let G represent } E_2; \\
\ldots \\
\text{let G represent } E_n;"
\]
may be replaced by
\[
\text{"let G represent } E_1, E_2, \ldots, E_n;".
\]

1.3.21 When $P_1,\ldots,P_n$ are <mark>'s and Z or $Z'$ is a <left priority>
or a <right priority> respectively,
\[
\text{"let } P_1 \text{ operate } ZZ'; \\
\text{let } P_2 \text{ operate } ZZ'; \\
\ldots \\
\text{let } P_n \text{ operate } ZZ';"
\]
may be replaced by
\[
\text{"let } P_1, P_2, \ldots, P_n \text{ operate } ZZ';".
\]

1.3.22 "before left" may be replaced by "before none left".
1.3.23 "after right" may be replaced by "after none right".
1.3.24 Let $n$ be an integer ($\geq 0$), $U_i$ be a <basic symbol> other than
\text{comment and silent for } i=1,2,\ldots,n.
"comment $U_1\ldots U_n$ silent" may be inserted between two symbols.

Syntax-10