Contents

Annexes

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

International Standard ISO/IEC 14977 was prepared by BSI (as BS 6154) and was adopted, under a special “fast-track procedure”, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by national bodies of ISO and IEC.

Annexes A and B of this International Standard are for information only.

Introduction

A syntactic metalanguage is an important tool of computer science. The concepts are well known, but many slightly different notations are in use. As a result syntactic metalanguages are still not widely used and understood, and the advantages of rigorous notations are unappreciated by many people.

Extended BNF brings some order to the formal definition of a syntax and will be useful not just for the definition of programming languages, but for many other formal definitions.

Since the definition of the programming language Algol 60 (Naur, 1960) the custom has been to define the syntax of a programming language formally. Algol 60 was defined with a notation now known as BNF or Backus-Naur Form. This notation has proved a suitable basis for subsequent languages but has frequently been extended or slightly altered. The many different notations are confusing and have prevented the advantages of formal unambiguous definitions from being widely appreciated. The syntactic metalanguage *Extended BNF* described in this standard is based on Backus-Naur Form and includes the most widely adopted extensions.

Syntactic metalanguages

A syntactic metalanguage is a notation for defining the syntax of a language by use of a number of rules. Each rule names part of the language (called a non-terminal symbol of the language) and then defines its possible forms. A terminal symbol of the language is an atom that cannot be split into smaller components of the language. A syntactic metalanguage is useful whenever a clear formal description and definition is required, e.g. the format for references in papers submitted to a journal, or the instructions for performing a complicated task.

A formal syntax definition has three distinct uses:

a) it names the various syntactic parts (i.e. non-terminal symbols) of the language;

b) it shows which sequences of symbols are valid sentences of the language;

c) it shows the syntactic structure of any sentence of the language.

The need for a standard syntactic metalanguage

Without a standard syntactic metalanguage every programming language definition starts by specifying the metalanguage used to define its syntax. This causes various problems:

Many different notations — It is unusual for two different programming languages to use the same metalanguage. Thus human readers are handicapped by having to learn a new metalanguage before they can study a new language.

Concepts not widely understood — The lack of a standard notation hinders the use of rigorous unambiguous definitions.

Imperfect notations — Because a metalanguage needs to be defined for every programming language, almost inevitably, the metalanguage contains defects. For example errors occurred in the drafting of RTL/2 (BS5904) and CORAL 66 (BS5905) because the metalanguages could not be typed easily.

Special purpose notations — A metalanguage defined for a particular programming language is often simplified by taking advantage of special features in the language to be defined. However, the metalanguage is then unsuitable for other programming languages.

Few general syntax processors — The multiplicity of syntactic metalanguages has limited the availability of computer programs to analyse and process syntaxes, e.g. to list a syntax neatly, to make an index of the symbols used in the syntax, to produce a syntax-checker for programs written in the language.

In practice experienced readers have little difficulty in picking up and learning a new notation, but even so the differences obscure mutual understanding and hinder communication. A standard metalanguage enables more people to crystallize vague ideas into an unambiguous definition. It is also useful because other people needing to provide formal definitions no longer need to reinvent similar concepts.

The objectives to be satisfied

It is desirable that a standard syntactic metalanguage should be:

a) concise, so that languages can be defined briefly and thus be more easily understood;

b) precise, so that the rules are unambiguous;

c) formal, so that the rules can be parsed, or otherwise processed, by a computer when required;

d) natural, so that the notation and format are relatively simple to learn and understand, even for those who are not themselves language designers; (The meaning of a symbol should not be surprising. It should also be possible to define the syntax of a language in a way that helps to indicate the meaning of the constructions.)

e) general, so that the notation is suitable for many purposes including the description of many different languages;

f) simple in its character set and with a notation that avoids, as far as is practicable, using characters that are not generally available on standard keyboards (both typewriters and computer terminals) so that the rules can be typed and can be processed by computer programs;

g) self describing, so that the notation is able to describe itself;

h) linear, so that the syntax can be expressed as a single stream of characters. (This simplifies printing a syntax. Computer processing of a syntax is also simpler.)

Some common syntactic metalanguages

Unfortunately none of the existing syntactic metalanguages was suitable for adoption as the standard, for example:

a) COBOL (ISO 1989:1985) lists alternatives vertically and uses brackets spreading over many lines. This is inconvenient for computer processing and cannot be prepared on typewriters.

b) Backus-Naur Form (used in ALGOL 60) has problems if the metasymbols `< > | ::=` occur in the language being defined. Some common forms of construction (e.g. comments) cannot be expressed naturally, other constructions (e.g. repetition) are long-winded.

c) The obsolete FORTRAN 77 (ISO 1539:1980) had 'railroad tracks'. These are easy to understand but difficult to prepare and to process on a computer or typewriter. The current version, FORTRAN 90 (ISO/IEC 1539:1991), no longer uses this notation.

Most other languages use a variant of one of these metalanguages. Most of them cannot be candidates for standardization because they use characters not in the language being defined as metasymbols of the metalanguage. This simplifies the metalanguage but prevents it from being used generally.

POSIX (ISO/IEC 9945-2:1993) includes two complementary facilities which both assume an ISO/IEC 646:1991 character set is applicable: LEX permits the definition and lexical analysis of regular expressions, but is inadequate for the description of an arbitrary context-free grammar, and YACC (Yet Another Compiler Compiler) is a parser generator for an LALR(1) grammar.

The standard metalanguage Extended BNF

Extended BNF, the metalanguage defined in this International Standard, is based on a suggestion by Niklaus Wirth (Wirth, 1977) that is based on Backus-Naur Form and that contains the most common extensions, i.e.:

a) Terminal symbols of the language are quoted so that any character, including one used in *Extended BNF*, can be defined as a terminal symbol of the language being defined.

b) `[` and `]` indicate optional symbols.

c) { and } indicate repetition.

d) Each rule has an explicit final character so that there is never any ambiguity about where a rule ends.

e) Brackets group items together. It is an obvious convenience to use (and) in their ordinary mathematical sense.

The main differences in *Extended BNF* are further features that experience has shown are often required when providing a formal definition:

a) *Defining an explicit number of items.* Fortran contains a rule that a label field contains exactly five characters; an identifier in PL/I or COBOL has up to 32 characters: rules such as these can be expressed only with difficulty in Backus-Naur Form. In practice, such definitions are often left incomplete and the rules qualified informally in English.

b) *Defining something by specifying the few exceptional cases.* An Algol **end**-comment ends at the first **end**, **else** or semicolon. A rule like this cannot be expressed concisely or clearly in Backus-Naur Form and is also usually specified informally in English.

c) *Including comments.* Programming languages and other structures with a complicated syntax need many rules to define them. The syntax will be clearer if explanations and cross-references can be provided; accordingly *Extended BNF* contains a comment facility so that ordinary text can be added to a syntax for the benefit of a human reader without affecting the formal meaning of the syntax.

d) *Meta-identifier.* A meta-identifier (the name of a non-terminal symbol in the language) need not be a single word or enclosed in brackets because there is an explicit concatenate symbol. This also ensures that the layout of a syntax (except in a terminal symbol) does not affect the language being defined.

e) *Extensions.* A user may wish to extend *Extended BNF*. A special-sequence is provided for this purpose, the format and meaning of which are not defined in the standard except to ensure that the start and end of an extension can always be seen easily. Various possible extensions are outlined in the following paragraphs.

Limitations and extensions

The main limitation of *Extended BNF* is that the language being defined needs to be linear, i.e. the symbols in a sentence of the language can be placed in an ordered sequence. For example knitting patterns and recipes in cooking are linear languages, but electric circuit diagrams are not.

A further limitation is that *Extended BNF* is inadequate for defining more complex forms of grammars. Such facilities were not provided because it was thought the main need was to define a notation sufficient for the simpler and commoner requirements.

Instead *Extended BNF* has been designed so that various extensions can be made in a natural way. There are two simple ways of extending the standard metalanguage. Firstly, the special-sequence concept provides a basic framework for any extension, the format between the special-sequence-characters being almost completely arbitrary. This method would be suitable for an action grammar, i.e. one specifying actions that are to take place as a sentence is parsed. Secondly, a meta-identifier can never be followed immediately by a left parenthesis in the standard metalanguage; thus another method of extending the metalanguage is to define the syntax and meaning of a meta-identifier followed by a sequence of parameters enclosed in parentheses. This would be reasonable in an attribute grammar where the rules ensure consistency between different parts of a sentence in the language being defined.

More complicated extensions are also possible. Annex A suggests how *Extended BNF* might be extended to define a two-level grammar.

Information technology — Syntactic metalanguage — Extended BNF

1 Scope

This International Standard defines a notation, *Extended BNF*, for specifying the syntax of a linear sequence of symbols. It defines both the logical structure of the notation and its graphical representation.

Extended BNF has applications in the definition of programming and other languages, as well as in other formal definitions, for example the commands to an operating system, or the precise format of data and results.

Examples of *Extended BNF* are given in clause 8.

NOTE — Like many other notations, *Extended BNF* can still be misused; thus it does not prevent someone from trying to define an unparsable or ambiguous language.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO 2382-15 : 1985, *Data processing — Vocabulary — Part 15: Programming languages.*

ISO/IEC 646 : 1991, *Information technology — ISO 7-bit coded character set for information interchange.*

ISO/IEC 6429 : 1992, *Information technology — Control functions for 7-bit and 8-bit coded character sets.*

BS 6154 : 1981, *Method of defining — Syntactic metalanguage.*

3 Definitions

For the purposes of this International Standard, the definitions given in ISO 2382-15 and the following definitions apply:

3.1 sequence: An ordered list of zero or more items.

3.2 subsequence: A sequence within a sequence.

3.3 non-terminal symbol: A syntactic part of the language being defined.

3.4 meta-identifier: The name of a non-terminal symbol.

3.5 start symbol: A non-terminal symbol that is defined by one or more syntax rules but does not occur in any other syntax rule.

3.6 sentence: A sequence of symbols that represents the start symbol.

3.7 terminal symbol: A sequence of one or more characters forming an irreducible element of a language.

NOTE — In this International Standard a terminal symbol of *Extended BNF* is called a terminal-character, and a terminal symbol of a language being defined by a syntax is represented by a terminal-string.

4 The form of each syntactic element of Extended BNF

NOTES

1 The following conventions are used:

a) Each meta-identifier of *Extended BNF* is written as one or more words joined together by hyphens;

b) A meta-identifier ending with "-symbol" is the name of a terminal symbol of *Extended BNF*.

2 The normal character representing each operator of *Extended BNF* and its implied precedence is (highest precedence at the top):

*	repetition-symbol
-	except-symbol
,	concatenate-symbol
|	definition-separator-symbol
=	defining-symbol
;	terminator-symbol

3 The normal precedence is over-ridden by the following bracket pairs:

'	first-quote-symbol	first-quote-symbol	'
"	second-quote-symbol	second-quote-symbol	"
(*	start-comment-symbol	end-comment-symbol	*)
(	start-group-symbol	end-group-symbol	)
[	start-option-symbol	end-option-symbol	]
{	start-repeat-symbol	end-repeat-symbol	}
?	special-sequence -symbol	special-sequence -symbol	?

4.1 General

The logical structure of *Extended BNF* is defined in 4.2 to 4.21.

4.2 Syntax

The syntax of a language consists of one or more syntax-rules.

4.3 Syntax-rule

A syntax-rule consists of a meta-identifier (the name of the non-terminal symbol being defined) followed by a defining-symbol followed by a definitions-list followed by a terminator-symbol.

4.4 Definitions-list

A definitions-list consists of an ordered list of one or more single-definitions separated from each other by a definition-separator-symbol.

4.5 Single-definition

A single-definition consists of an ordered list of one or more syntactic-terms separated from each other by a concatenate-symbol.

4.6 Syntactic-term

A syntactic-term consists of either:

a) a syntactic-factor, or

b) a syntactic-factor followed by an except-symbol followed by a syntactic-exception.

4.7 Syntactic exception

A syntactic-exception consists of a syntactic-factor subject to the restriction that the sequences of symbols represented by the syntactic-exception could equally be represented by a syntactic-factor containing no meta-identifiers.

NOTE — If a syntactic-exception is permitted to be an arbitrary syntactic-factor, *Extended BNF* could define a wider class of languages than the context-free grammars, including attempts which lead to Russell-like paradoxes, e.g.

```
xx = "A" - xx;
```

Is `"A"` an example of `xx`? Such licence is undesirable and the form of a syntactic-exception is therefore restricted to cases that can be proved to be safe. Thus whereas a syntactic-factor is in general equivalent to some context-free grammar, a syntactic-exception is always equivalent to some regular grammar. It may be shown that the difference between a context-free grammar and a regular grammar is always another context-free grammar; hence a syntactic-term (and hence any grammar defined according to this standard) is equivalent to some context-free grammar.

4.8 Syntactic-factor

A syntactic-factor consists of either:

a) an integer followed by a repetition-symbol followed by a syntactic-primary, or

b) a syntactic-primary.

4.9 Integer

An integer consists of an ordered list of one or more decimal-digits.

4.10 Syntactic-primary

A syntactic-primary consists of one of the following:

a) an optional-sequence;

b) a repeated-sequence;

c) a grouped-sequence;

d) a meta-identifier;

e) a terminal-string;

f) a special-sequence;

g) an empty-sequence.

4.11 Optional-sequence

An optional-sequence consists of a start-option-symbol followed by a definitions-list followed by an end-option-symbol.

4.12 Repeated sequence

A repeated-sequence consists of a start-repeat-symbol followed by a definitions-list followed by an end-repeat-symbol.

4.13 Grouped sequence

A grouped-sequence consists of a start-group-symbol followed by a definitions-list followed by an end-group-symbol.

4.14 Meta-identifier

A meta-identifier consists of an ordered list of one or more meta-identifier-characters subject to the condition that the first meta-identifier-character is a letter.

4.15 Meta-identifier-character

A meta-identifier-character is a letter or a decimal-digit.

4.16 Terminal-string

A terminal-string consists of either:

a) A first-quote-symbol followed by a sequence of one or more first-terminal-characters followed by a first-quote-symbol, or

b) A second-quote-symbol, followed by a sequence of one or more second-terminal-characters followed by a second-quote-symbol.

4.17 First-terminal-character

A first-terminal-character is any terminal-character except a first-quote-symbol.

4.18 Second-terminal-character

A second-terminal-character is any terminal-character except a second-quote-symbol.

4.19 Special-sequence

A special-sequence consists of a special-sequence-symbol followed by a (possibly empty) sequence of special-sequence-characters followed by a special-sequence-symbol.

4.20 Special-sequence-character

A special-sequence-character is any terminal-character except a special-sequence-symbol.

4.21 Empty-sequence

An empty-sequence consists of the empty sequence of terminal-characters.

4.22 Further examples

The following example is a syntax-rule that states that a Fortran 77 continuation line starts with 5 blanks, the sixth character must not be a blank or zero, and there must not be more than 72 (= 5+1+66) characters altogether.

```
Fortran 77 continuation line = 5 * " ",
   (character - (" " | "0")), 66 * [character] ;
```

In Fortran 66, the definition of a continuation line is more complicated. The following example is a syntax-rule that states that a continuation line must not start with `C`, there must be at least 6 characters, the sixth character must not be a blank or zero, and there must not be more than 72 (= 1+4+1+66) characters altogether.

```
Fortran 66 continuation line = character - "C",
   4 * character, character - (" " | "0"),
   66 * [character] ;
```

5 The symbols represented by each syntactic element

5.1 General

Each syntax-rule is a syntax rule that defines (possibly empty) sequences of terminal and non-terminal symbols. Each of these sequences of symbols is represented by the non-terminal symbol named by the meta-identifier at the start of the syntax-rule. 5.2 to 5.12 define the sequences of symbols that are represented by any definitions-list.

NOTES

1 When the syntax of a complete language is defined there is:

a) a start symbol, and

b) at least one syntax-rule starting with each meta-identifier used as a syntactic-primary.

2 It is more difficult to understand a language if there are several syntax-rules defining a meta-identifier and no indication that each definition only partly defines the non-terminal symbol.

5.2 Terminal-string

A terminal-string represents either:

a) the sequence of first-terminal-characters between its first-quote-symbols, or

b) the sequence of second-terminal-characters between its second-quote-symbols.

5.3 Meta-identifier

A meta-identifier used as a syntactic-primary represents any sequence of symbols defined by the definitions-list of any syntax-rule that starts with that meta-identifier.

5.4 Grouped-sequence

A grouped-sequence represents any sequence of symbols defined by the definitions-list enclosed by its start-group-symbol and end-group-symbol.

5.5 Optional-sequence

An optional-sequence represents either:

a) the empty sequence of symbols, or

b) any sequence of symbols defined by the definitions-list enclosed by its start-option-symbol and end-option-symbol.

5.6 Repeated-sequence

A repeated-sequence represents a (possibly empty) sequence of subsequences where each subsequence is any sequence of symbols defined by the definitions-list enclosed by the start-repeat-symbol and end-repeat-symbol.

5.7 Syntactic-factor

A syntactic-factor represents an explicit number of subsequences where each subsequence is a sequence of symbols represented by the syntactic-primary that is part of that syntactic-factor. The required number of subsequences equals one when no integer is given and otherwise is equal to the value of the integer.

As examples the following syntax-rules illustrate the facilities for expressing repetition.

```
aa = "A";
bb = 3 * aa, "B";
cc = 3 * [aa], "C";
dd = {aa}, "D";
ee = aa, {aa}, "E";
ff = 3 * aa, 3 * [aa], "F";
gg = 3 * {aa}, "D";
```

Terminal-strings defined by these rules are as follows:

```
aa:  A
bb:  AAAB
cc:  C   AC   AAC   AAAC
dd:  D   AD   AAD   AAAD   AAAAD   etc.
ee:  AE   AAE   AAAE   AAAAE   AAAAAE   etc.
ff:  AAAF   AAAAF   AAAAAF   AAAAAAF
```

NOTE — The definition for `gg`, although syntactically valid, is not sensible. The sequences of symbols represented by `gg` are identical with those given by `dd` but cannot be parsed unambiguously.

5.8 Syntactic-term

When a syntactic-term is a single syntactic-factor it represents any sequence of symbols represented by that syntactic-factor.

When a syntactic-term is a syntactic-factor followed by an except-symbol followed by a syntactic-exception it represents any sequence of symbols that satisfies both of the conditions:

a) it is a sequence of symbols represented by the syntactic-factor,

b) it is not a sequence of symbols represented by the syntactic-exception.

As examples the following syntax-rules illustrate the facilities provided by the except-symbol.

```
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";
vowel = "A" | "E" | "I" | "O" | "U";
consonant = letter - vowel;
ee = {"A"}-, "E";
```

Terminal-strings defined by these rules are as follows:

```
letter:     A  B  C  D  E  F  G  H  I  J  etc.
vowel:      A  E  I  O  U
consonant:   B  C  D  F  G  H  J  K  L  M  etc.
ee:        AE  AAE  AAAE  AAAAE  AAAAAE  etc.
```

NOTE — {"A"}- represents a sequence of one or more A's because it is a syntactic-term with an empty syntactic-exception.

5.9 Single-definition

A single-definition represents a sequence of one or more subsequences where each subsequence is a sequence of symbols represented by the corresponding syntactic-term in that single-definition.

5.10 Definitions-list

A definitions-list represents any sequence of symbols that is represented by any one of the single-definitions forming that definitions-list.

5.11 Special-sequence

The sequence of symbols represented by a special-sequence is outside the scope of this International Standard. Only the format of a special-sequence is defined in this International Standard. A special-sequence provides a notation for extensions which a user may require.

5.12 Empty-sequence

An empty-sequence represents the empty sequence of symbols.

6 Layout and Comments

6.1 General

The layout of the syntax on a page is almost completely arbitrary. 6.2 to 6.4 define that a non-printing character such as space or new-line has no formal effect on a syntax if the character is outside a terminal-string or pair of characters forming a single terminal-character. 6.5 to 6.7 define where arbitrary text may be inserted as a comment in a syntax.

NOTES

1 It is much easier for a person to read and understand a syntax if each syntax-rule starts on a new line and the various metalanguage symbols are sensibly spaced.

2 A language defined by *Extended BNF* may have completely different lexical rules from *Extended BNF* itself.

3 Comments enable explanatory text to be added to a syntax and thus help a human to understand a syntax. For example, syntax-rules can be numbered and each meta-identifier followed by a comment identifying the position of the syntax-rule that defines it. It is recommended that any comment concerning a syntax-rule should appear before the terminator-symbol of the rule.

4 Comments have no formal effect on the language defined by a syntax.

6.2 Terminal-character

A terminal-character of *Extended BNF* is one of the following:

a) a letter;

b) a decimal-digit;

c) a concatenate-symbol;

d) a defining-symbol;

e) a definition-separator-symbol;

f) an end-comment-symbol;

g) an end-group-symbol;

h) an end-option-symbol;

i) an end-repeat-symbol;

j) an except-symbol;

k) a first-quote-symbol;

l) a repetition-symbol;

m) a second-quote-symbol;

n) a special-sequence-symbol;

o) a start-comment-symbol;

p) a start-group-symbol;

q) a start-option-symbol;

r) a start-repeat-symbol;

s) a terminator-symbol;

t) an other-character.

6.3 Gap-free-symbol

A gap-free-symbol is either:

a) a terminal-character that is neither a first-quote-symbol nor a second-quote-symbol, or

b) a terminal-string.

6.4 Gap-separator

A gap-separator is one of the non-printing characters: space, horizontal-tabulation, new-line, vertical-tabulation, or form-feed.

One or more gap-separators may be placed:

a) before a syntax, and

b) between any two gap-free-symbols of a syntax, and

c) after a syntax

without affecting the language defined by the syntax.

6.5 Commentless-symbol

A commentless-symbol is one of the following:

a) a terminal-character that is neither a letter nor a decimal-digit nor a first-quote-symbol nor a second-quote-symbol nor a start-comment-symbol nor an end-comment-symbol nor a special-sequence-symbol nor an other-character;

b) a meta-identifier;

c) an integer;

d) a terminal-string;

e) a special-sequence.

6.6 Comment-symbol

A comment-symbol is one of the following:

a) a bracketed-textual-comment;

b) a commentless-symbol;

c) an other-character.

6.7 Bracketed-textual-comment

A bracketed-textual-comment is a start-comment-symbol followed by a (possibly empty) sequence of comment-symbols followed by an end-comment-symbol.

One or more bracketed-textual-comments may be placed:

a) before a syntax, and

b) between any two commentless-symbols of a syntax, and

c) after a syntax

without affecting the language defined by the syntax.

NOTE — 6.5 to 6.7 imply that bracketed-textual-comments cannot appear in any of the following:

a) a meta-identifier;

b) an integer;

c) a special-sequence;

d) a terminal-string.

7 The representation of each terminal-character in Extended BNF

7.1 General

The representation of each terminal-character and gap-separator in *Extended BNF* using the characters in the 7-bit character set (ISO/IEC 646:1991 International Reference Version) is defined in 7.2 to 7.8.

7.2 Letters and digits

Each letter and decimal-digit is represented by the corresponding character.

7.3 Other terminal characters

Table 1 defines the character representation for each terminal-character that is neither a letter, nor a decimal-digit nor an other-character.

7.4 Alternative representations

Table 2 defines alternative character representations for some terminal-characters.

Table 1 — Representation of terminal-characters

Metalanguage symbol	Normal representation
concatenate-symbol	, comma
defining-symbol	= equals sign
definition-separator-symbol	| vertical line
end-comment-symbol	*) asterisk, right parenthesis
end-group-symbol	) right parenthesis
end-option-symbol	] right square bracket
end-repeat-symbol	} right curly bracket
except-symbol	- hyphen-minus
first-quote-symbol	´ apostrophe
repetition-symbol	* asterisk
second-quote-symbol	" quotation mark
special-sequence-symbol	? question mark
start-comment-symbol	(* left parenthesis, asterisk
start-group-symbol	(left parenthesis
start-option-symbol	[left square bracket
start-repeat-symbol	{ left curly bracket
terminator-symbol	; semicolon

Table 2 — Alternative representation of terminal-characters

Metalanguage symbol	Alternative representation
definition-separator-symbol	/ solidus
definition-separator-symbol	! exclamation mark
end-option-symbol	/) solidus, right parenthesis
end-repeat-symbol	:) colon, right parenthesis
start-option-symbol	(/ left parenthesis, solidus
start-repeat-symbol	(: left parenthesis, colon
terminator-symbol	. full stop

NOTES

1 The main reason for specifying alternative representations is that not all computers and typewriters have the characters listed in table 1.

2 To avoid confusion, the representation of a terminal-character in any one document should be consistent.

3 7.2 to 7.4 imply that the characters required for *Extended BNF* are:

letters digits = , - * ( ) ?
| or / or !
/ or both of [ ]
: or both of { }
´ or " (Both characters are needed if either is a terminal symbol of the language being defined)

7.5 Other-character

An other-character is any other character in the ISO/IEC 646:1991 character set that is neither:

a) a control character, nor

b) required to represent any other terminal-character.

NOTE — When the terminal-characters are represented as specified in table 1, the other-characters are:

space
. full stop
: colon
! exclamation mark
+ plus sign
_ lowline
% percent sign
@ commercial at
& ampersand
# number sign
$ dollar sign
< less-than sign
> greater-than sign
/ solidus
\ reverse solidus
^ circumflex accent
` grave accent
~ tilde

7.6 Gap-separator

A gap-separator is represented as follows:

a) a space is represented by a Space character,

b) a horizontal-tabulation is represented by a Horizontal Tabulation character,

Table 3 — Character pairs that represent a single terminal-character

(*
*)
(:
:)
(/
/)

Table 4 — Invalid sequences of characters

(*)
(:)
(/)

c) a new-line is represented by a (possibly empty) sequence of Carriage Return characters, a Line Feed character, and a (possibly empty) sequence of Carriage Return characters,

d) a vertical-tabulation is represented by a Vertical Tabulation character,

e) a form-feed is represented by a Form Feed character.

7.7 Terminal-characters represented by a pair of characters

Each pair of characters in table 3 always represents a single terminal-character in a syntax-rule except inside a terminal-string or special-sequence.

NOTE — This restriction is necessary because these character sequences are ambiguous, for example `/)` could be a definition-separator-symbol followed by an end-group-symbol, or an end-option-symbol.

7.8 Invalid character sequences

Each line of table 4 specifies a character sequence that does not appear in a syntax-rule outside a terminal-string or special-sequence.

NOTE — This restriction is necessary because these character sequences are ambiguous, for example `(*)` could be a start-comment-symbol followed by an end-group-symbol, or a start-group-symbol followed by an end-comment-symbol.

Inserting a gap-separator allows either meaning, for example `(*)` is a start-comment-symbol followed by an end-group-symbol, and `( *)` is a start-group-symbol followed by an end-comment-symbol.

8 Examples

8.1 The syntax of Extended BNF

```
(*
  The syntax of Extended BNF can be defined using
  itself. There are four parts in this example,
  the first part names the characters, the second
  part defines the removal of unnecessary non-
  printing characters, the third part defines the
  removal of textual comments, and the final part
  defines the structure of Extended BNF itself.

  Each syntax rule in this example starts with a
  comment that identifies the corresponding clause
  in the standard.

  The meaning of special-sequences is not defined
  in the standard. In this example (see the
  reference to 7.6) they represent control
  functions defined by ISO/IEC 6429:1992.
  Another special-sequence defines a
  syntactic-exception (see the reference to 4.7).
*)

(*
  The first part of the lexical syntax defines the
  characters in the 7-bit character set (ISO/IEC
  646:1991) that represent each terminal-character
  and gap-separator in Extended BNF.
*)
(* see 7.2 *) 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'
  | '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';
(* see 7.2 *) decimal digit
  = '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7'
  | '8' | '9';
(*
  The representation of the following
  terminal-characters is defined in clauses 7.3,
  7.4 and tables 1, 2.
*)
concatenate symbol = ',';
defining symbol = '=';
definition separator symbol = '|' | '/' | '!';
end comment symbol = '*)';
end group symbol = ')';
end option symbol = ']' | '/)';
end repeat symbol = '}' | ':)';
except symbol = '-';
first quote symbol = "'";
repetition symbol = '*';
second quote symbol = '"';
special sequence symbol = '?';
start comment symbol = '(*';
start group symbol = '(';
start option symbol = '[' | '(/';
start repeat symbol = '{' | '(:';
terminator symbol = ';' | '.';
(* see 7.5 *) other character
  = ' ' | ':' | '+' | '_' | '%' | '@'
  | '&' | '#' | '$' | '<' | '>' | '\'
  | '^' | '`' | '~';
(* see 7.6 *) space character = ' ';
```

```
horizontal tabulation character
  = ? ISO 6429 character Horizontal Tabulation ? ;
new line
  = { ? ISO 6429 character Carriage Return ? },
  ? ISO 6429 character Line Feed ?,
  { ? ISO 6429 character Carriage Return ? };
vertical tabulation character
  = ? ISO 6429 character Vertical Tabulation ? ;
form feed
  = ? ISO 6429 character Form Feed ? ;

(*
  The second part of the syntax defines the
  removal of unnecessary non-printing characters
  from a syntax.
*)
(* see 6.2 *) terminal character
  = letter
  | decimal digit
  | concatenate symbol
  | defining symbol
  | definition separator symbol
  | end comment symbol
  | end group symbol
  | end option symbol
  | end repeat symbol
  | except symbol
  | first quote symbol
  | repetition symbol
  | second quote symbol
  | special sequence symbol
  | start comment symbol
  | start group symbol
  | start option symbol
  | start repeat symbol
  | terminator symbol
  | other character;
(* see 6.3 *) gap free symbol
  = terminal character
    - (first quote symbol | second quote symbol)
  | terminal string;
(* see 4.16 *) terminal string
  = first quote symbol, first terminal character,
    {first terminal character},
    first quote symbol
  | second quote symbol, second terminal character,
    {second terminal character},
    second quote symbol;
(* see 4.17 *) first terminal character
  = terminal character - first quote symbol;
(* see 4.18 *) second terminal character
  = terminal character - second quote symbol;
(* see 6.4 *) gap separator
  = space character
  | horizontal tabulation character
  | new line
  | vertical tabulation character
  | form feed;
(* see 6.5 *) syntax
  = {gap separator},
  gap free symbol, {gap separator},
  {gap free symbol, {gap separator}};

(*
  The third part of the syntax defines the
  removal of bracketed-textual-comments from
  gap-free-symbols that form a syntax.
*)
(* see 6.6 *) commentless symbol
  = terminal character
    - (letter
       | decimal digit
       | first quote symbol
       | second quote symbol
       | start comment symbol
       | end comment symbol
       | special sequence symbol
       | other character)
  | meta identifier
  | integer
  | terminal string
  | special sequence;
(* see 4.9 *) integer
  = decimal digit, {decimal digit};
(* see 4.14 *) meta identifier
  = letter, {meta identifier character};
(* see 4.15 *) meta identifier character
  = letter
  | decimal digit;
(* see 4.19 *) special sequence
  = special sequence symbol,
    {special sequence character},
    special sequence symbol;
(* see 4.20 *) special sequence character
  = terminal character - special sequence symbol;
(* see 6.7 *) comment symbol
  = bracketed textual comment
  | other character
  | commentless symbol;
(* see 6.8 *) bracketed textual comment
  = start comment symbol, {comment symbol},
    end comment symbol;
(* see 6.9 *) syntax
  = {bracketed textual comment},
    commentless symbol,
    {bracketed textual comment},
    {commentless symbol,
      {bracketed textual comment}};

(*
  The final part of the syntax defines the
  abstract syntax of Extended BNF, i.e. the
  structure in terms of the commentless symbols.
*)

(* see 4.2 *) syntax
  = syntax rule, {syntax rule};
(* see 4.3 *) syntax rule
  = meta identifier, defining symbol,
    definitions list, terminator symbol;
(* see 4.4 *) definitions list
  = single definition,
    {definition separator symbol,
      single definition};
(* see 4.5 *) single definition
  = syntactic term,
    {concatenate symbol, syntactic term};
(* see 4.6 *) syntactic term
  = syntactic factor,
    [except symbol, syntactic exception];
(* see 4.7 *) syntactic exception
  = ? a syntactic-factor that could be replaced
      by a syntactic-factor containing no
      meta-identifiers
    ?  ;
(* see 4.8 *) syntactic factor
  = [integer, repetition symbol],
    syntactic primary;
(* see 4.10 *) syntactic primary
  = optional sequence
  | repeated sequence
  | grouped sequence
```

```
  | meta identifier
  | terminal string
  | special sequence
  | empty sequence;
(* see 4.11 *) optional sequence
  = start option symbol, definitions list,
  end option symbol;
(* see 4.12 *) repeated sequence
  = start repeat symbol, definitions list,
  end repeat symbol;
(* see 4.13 *) grouped sequence
  = start group symbol, definitions list,
  end group symbol;
(* see 4.21 *) empty sequence
  = ;
```

8.2 Extended BNF used to define itself informally

```
(*
  This example defines Extended BNF
  informally. Many of the syntax rules include
  a comment to explain their meaning; inside a
  comment a meta identifier is enclosed in angle
  brackets  <  and  >  to avoid confusion with
  similar English words. The non-terminal symbols
  <letter>, <decimal digit> and <character> are
  not defined.  The position of <comments> is
  stated in a comment but not formally defined.
*)
syntax = syntax rule, {syntax rule};
syntax rule
  = meta identifier, '=', definitions list, ';'
  (* A <syntax rule> defines the sequences of
    symbols represented by a <meta identifier> *);
definitions list
  = single definition, {'|', single definition}
  (* |  separates alternative
    <single definitions>  *);
single definition = term, {',', term}
  (* ,  separates successive <terms>  *);
term = factor, ['-', exception]
  (* A <term> represents any sequence of symbols
    that is defined by the <factor> but
    not defined by the <exception>  *);
exception =  factor
  (* A <factor> may be used as an <exception>
    if it could be replaced by a <factor>
    containing no <meta identifiers>   *);
factor = [integer, '*'], primary
  (* The <integer> specifies the number of
    repetitions of the <primary>  *);
primary
  = optional sequence | repeated sequence
  | special sequence | grouped sequence
  | meta identifier | terminal string | empty;
empty = ;
optional sequence = '[', definitions list, ']'
  (* The brackets  [  and  ]  enclose symbols
    which are optional  *);
repeated sequence = '{', definitions list, '}'
  (* The brackets  {  and  }  enclose symbols
    which may be repeated any number of times  *);
grouped sequence = '(', definitions list, ')'
  (* The brackets  (  and  )  allow any
    <definitions list> to be a <primary>  *);
terminal string
  = "'", character - "'", {character - "'"}, "'"
  | '"', character - '"', {character - '"'}, '"'
  (* A <terminal string> represents the
    <characters> between the quote symbols
    '_'  or  "_"  *);
meta identifier = letter, {letter | decimal digit}
  (* A <meta identifier> is the name of a
    syntactic element of the language being
    defined  *);
integer = decimal digit, {decimal digit};
special sequence = '?', {character - '?'}, '?'
  (* The meaning of a <special sequence> is not
    defined in the standard metalanguage.  *);
comment = '(*', {comment symbol}, '*)'
  (* A comment is allowed anywhere outside a
    <terminal string>, <meta identifier>,
    <integer> or <special sequence> *);
comment symbol
  = comment | terminal string | special sequence
  | character;
```

8.3 Extended BNF defined informally

```
(*
  THIS EXAMPLE USES THE REPRESENTATION DEFINED
  IN TABLE 2.
*)
SYNTAX = SYNTAX RULE, (: SYNTAX RULE :).
SYNTAX RULE
  = META IDENTIFIER, '=', DEFINITIONS LIST, '.'.
DEFINITIONS LIST
  = SINGLE DEFINITION,
    (: '/', SINGLE DEFINITION :).
SINGLE DEFINITION = TERM, (: ',', TERM :).
TERM = FACTOR, (/ '-', EXCEPTION /).
EXCEPTION = FACTOR.
FACTOR = (/ INTEGER, '*' /), PRIMARY.
PRIMARY
  = OPTIONAL SEQUENCE / REPEATED SEQUENCE
  / SPECIAL SEQUENCE / GROUPED SEQUENCE
  / META IDENTIFIER / TERMINAL / EMPTY.
EMPTY = .
OPTIONAL SEQUENCE = '(/', DEFINITIONS LIST, '/)'.
REPEATED SEQUENCE = '(:', DEFINITIONS LIST, ':)'.
GROUPED SEQUENCE = '(', DEFINITIONS LIST, ')'.
TERMINAL
  = "'", CHARACTER - "'",
    (: CHARACTER - "'" :), "'"
  / '"', CHARACTER - '"',
    (: CHARACTER - '"' :), '"'.
META IDENTIFIER = LETTER, (: LETTER / DIGIT :).
INTEGER = DIGIT, (: DIGIT :).
SPECIAL SEQUENCE = '?', (: CHARACTER - '?' :), '?'.
COMMENT = '(*', (: COMMENT SYMBOL :), '*)'.
COMMENT SYMBOL
  = COMMENT / TERMINAL / SPECIAL SEQUENCE
    / CHARACTER.
```

Annex A
(informative)
Two-level grammars

A.1 For most users, the facilities described in this International Standard will be more than adequate. However some users will want to make more powerful extensions. This annex illustrates the possibilities by suggesting how *Extended BNF* might be extended to define a two-level grammar. This sort of grammar, used for example in Algol 68 (van Wijngaarden, 1975) provides a more precise but less direct method of defining languages. Although the notation (also known as a van Wijngaarden grammar, or a W-grammar) is more powerful, it is more complicated and, as the authors of Algol 68 recognized, "may be difficult for the uninitiated reader".

There are two sorts of rules in a two-level grammar. Some, called hyper-rules, are similar to the syntax-rules in *Extended BNF* except that they may include special words known as metanotions. Other rules, called metaproduction rules, define the sequences of symbols that correspond to each metanotion. The syntax-rules of a language are generated by appropriately replacing each metanotion in a hyper-rule. When a metanotion occurs more than once in a hyper-rule, identical replacements for it are made when generating a syntax-rule. Metanotions inside terminal strings and comments are also systematically replaced.

A.2 Little extra notation is needed to extend *Extended BNF* into a two-level grammar:

a) Introduce a metaproduction-defining-symbol, e.g. == so that a metaproduction rule can be distinguished from a hyper-rule.

b) Distinguish metanotions from other hypernotions by using upper-case letters for metanotions, and lower-case letters for hypernotions and meta-identifiers.

For example, the language defined by the two-level grammar:

```
metaproduction rule:
  INTREAL == integer | real;
hyper-rules:
  program = {statement}, 'end';
  statement = INTREAL statement;
  INTREAL statement
    = 'print INTREAL', INTREAL expression;
  INTREAL expression = INTREAL value,
    {('+' | '-' | '*' | '/'), INTREAL value};
  integer value = digit, {digit};
  real value
    = digit, '.', digit, {digit}, '@', digit;
```

is equivalent to the language (defined using *Extended BNF*):

```
program = {statement}, 'end';
statement = integer statement;
statement = real statement;
integer statement
  = 'print integer', integer expression;
real statement = 'print real', real expression;
integer expression = integer value,
  {('+' | '-' | '*' | '/'), integer value};
real expression = real value,
  {('+' | '-' | '*' | '/'), real value};
integer value = digit, {digit};
real value
  = digit, '.', digit, {digit}, '@', digit;
```

A.3 The syntax of *Extended BNF* would need to be altered as follows:

a) Insert the following additional rules:

```
metaproduction rule
  = metanotion, metaproduction defining symbol,
  hypernotion,
  {definition separator symbol, hypernotion},
  terminator symbol;
metanotion
  = upper case letter, {upper case letter};
metaproduction defining symbol = "==";
hypernotion
  = letter, {letter | decimal digit};
upper case 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';
lower case 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';
```

b) Alter the existing rules:

```
syntax = hyper rule, {hyper rule};
hyper rule = hypernotion, defining symbol,
  definitions list, terminator symbol;
syntactic primary
  = optional sequence | repeated sequence
  | grouped sequence | hypernotion
  | terminal string | special sequence
  | empty sequence;
letter = upper case letter | lower case letter;
meta identifier
  = lower case letter,
    {lower case letter | decimal digit};
```

However, this simple definition would leave some problems unresolved: the substitution for a metanotion may not be uniquely defined, there may be an infinite number of production rules, and there may be production rules of infinite length.

Annex B
(informative)
Bibliography

The following standards and papers are referred to only in the introduction.

ISO 1539:1980, *Endorsement of ANSI X3.9-1978,* American National Standard — Programming Language FORTRAN. *American National Standards Institute, New York, USA. 1978.*

ISO/IEC 1539:1991, *Information technology — Programming languages — FORTRAN.*

ISO 1989:1985, *Endorsement of ANSI X3.23-1985,* American National Standard — Programming Language COBOL. *American National Standards Institute, New York, USA. 1985.*

ISO/IEC 9945-2 : 1993, *Information technology — Portable Operating System Interface (POSIX) — Part 2: Shell and utilities.*

BS 5904, *Programming languages - RTL/2, British Standards Institution, 1979.*

BS 5905, *Programming languages - CORAL 66, British Standards Institution, 1979.*

(Naur, 1960), *P Naur (Editor), Revised Report on the Algorithmic Language ALGOL 60, Computer Journal, Vol 5, No 4, pp349-367, Jan 1963.*

(van Wijngaarden, 1975), *A van Wijngaarden, B J Mailloux, J E L Peck, C H A Koster, M Sintzoff, C H Lindsey, L G L T Meertens, R G Fisker, Revised report on the Algorithmic Language ALGOL 68, Acta Informatica, Vol 5, parts 1-3, 1975 (also published in SIGPLAN Notices, Vol 12, No 5, pp1-70, May 1977).*

(Wirth, 1977), *N Wirth, What can we do about the unnecessary diversity of notation for syntactic definitions? Comm ACM, Vol 20, No 11, Nov 1977, p822.*