WIKIPEDIA
The Free Encyclopedia
WIKIPEDIA

Extended Backus–Naur form

In computer science, **extended Backus–Naur form** (**EBNF**) is a family of metasyntax notations, any of which can be used to express a context-free grammar. EBNF is used to make a formal description of a formal language such as a computer programming language. They are extensions of the basic Backus–Naur form (BNF) metasyntax notation. The earliest EBNF was developed by Niklaus Wirth, incorporating some of the concepts (with a different syntax and notation) from Wirth syntax notation. Today, many variants of EBNF are in use. The International Organization for Standardization adopted an EBNF Standard, ISO/IEC 14977, in 1996. [1] [2] According to Zaytsev, however, this standard "only ended up adding yet another three dialects to the chaos" and, after noting its lack of success, also notes that the ISO EBNF is not even used in all ISO standards. [3] Wheeler argues against using the ISO standard when using an EBNF and recommends considering alternative EBNF notations such as the one from the W3C Extensible Markup Language (XML) 1.0 (Fifth Edition). [4] This article uses EBNF as specified by the ISO for examples applying to all EBNFs. Other EBNF variants use somewhat different syntactic conventions.

Basics

EBNF is a code that expresses the syntax of a formal language.[5] An EBNF consists of terminal symbols and non-terminal production rules which are the restrictions governing how terminal symbols can be combined into a valid sequence. Examples of terminal symbols include alphanumeric characters, punctuation marks, and whitespace characters.

The EBNF defines production rules where sequences of symbols are respectively assigned to a nonterminal:

```
digit excluding zero = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
digit                = "0" | digit excluding zero ;
```

This production rule defines the nonterminal *digit* which is on the left side of the assignment. The vertical bar represents an alternative and the terminal symbols are enclosed with quotation marks followed by a semicolon as terminating character. Hence a *digit* is a *0* or a *digit excluding zero* that can be *1* or *2* or *3* and so forth until *9*.

A production rule can also include a sequence of terminals or nonterminals, each separated by a comma:

```
twelve                          = "1", "2" ;
two hundred one                 = "2", "0", "1" ;
three hundred twelve            = "3", twelve ;
twelve thousand two hundred one = twelve, two hundred one ;
```

Expressions that may be omitted or repeated can be represented through curly braces { ... }:

```
positive integer = digit excluding zero, { digit } ;
```

In this case, the strings *1*, *2*, ..., *10*, ..., *10000*, ... are correct expressions. To represent this, everything that is set within the curly braces may be repeated arbitrarily often, including not at all.

An option can be represented through squared brackets [...]. That is, everything that is set within the square brackets may be present just once, or not at all:

```
integer = "0" | [ "-" ], positive integer ;
```

Therefore, an integer is a zero (*0*) or a positive integer that may be preceded by an optional minus sign.

EBNF also provides, among other things, the syntax to describe repetitions (of a specified number of times), to exclude some part of a production, and to insert comments in an EBNF grammar.

Table of symbols

The following represents a proposed ISO/IEC 14977 standard, by R. S. Scowen, page 7, tables 1 and 2.

Usage	Notation	Alternative	Meaning
definition	`=`		
concatenation	`,`		
termination	`;`	`.`	
alternation	`|`	`/` or `!`	
optional	`[ ... ]`	`(/ ... /)`	none or once
repetition	`{ ... }`	`(: ... :)`	none or more
grouping	`( ... )`		
terminal string	`" ... "`	`' ... '`	
comment	`(* ... *)`		
special sequence	`? ... ?`		
exception	`-`		

Examples

Syntax diagram

EBNF

Even EBNF can be described using EBNF. Consider below grammar (using conventions such as "-" to indicate set disjunction, "+" to indicate one or more matches, and "?" for optionality):

```
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" ;

digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;

symbol = "[" | "]" | "{" | "}" | "(" | ")" | "<" | ">"
       | "'" | '"' | "=" | "|" | "." | "," | ";" | "-"
       | "+" | "*" | "?" | "\n" | "\t" | "\r" | "\f" | "\b" ;

character = letter | digit | symbol | "_" | " " ;
identifier = letter , { letter | digit | "_" } ;

S = { " " | "\n" | "\t" | "\r" | "\f" | "\b" } ;

terminal = "'" , character - "'" , { character - "'" } , "'"
         | '"' , character - '"' , { character - '"' } , '"' ;

terminator = ";" | "." ;

term = "(" , S , rhs , S , ")"
     | "[" , S , rhs , S , "]"
     | "{" , S , rhs , S , "}"
     | terminal
     | identifier ;

factor = term , S , "?"
       | term , S , "*"
       | term , S , "+"
       | term , S , "-" , S , term
       | term , S ;

concatenation = ( S , factor , S , "," ? ) + ;
alternation = ( S , concatenation , S , "|" ? ) + ;

rhs = alternation ;
lhs = identifier ;

rule = lhs , S , "=" , S , rhs , S , terminator ;

grammar = ( S , rule , S ) * ;
```

Pascal

A Pascal-like programming language that allows only assignments can be defined in EBNF as follows:

```
(* a simple program syntax in EBNF - Wikipedia *)
program = 'PROGRAM', white space, identifier, white space,
           'BEGIN', white space,
           { assignment, ";", white space },
           'END.' ;
identifier = alphabetic character, { alphabetic character | digit } ;
number = [ "-" ], digit, { digit } ;
string = '"' , { all characters - '"' }, '"' ;
assignment = identifier , ":=" , ( number | identifier | string ) ;
alphabetic character = "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" ;
white space = ? white space characters ? ;
all characters = ? all visible characters ? ;
```

For example, a syntactically correct program then could be:

```
PROGRAM DEMO1
BEGIN
  A:=3;
  B:=45;
  H:=-100023;
  C:=A;
  D123:=B34A;
  BABOON:=GIRAFFE;
  TEXT:="Hello world!";
END.
```

The language can easily be extended with control flows, arithmetical expressions, and Input/Output instructions. Then a small, usable programming language would be developed.

Advantages over BNF

Any grammar defined in EBNF can also be represented in BNF, though representations in the latter are generally lengthier. E.g., options and repetitions cannot be directly expressed in BNF and require the use of an intermediate rule or alternative production defined to be either nothing or the optional production for option, or either the repeated production of itself, recursively, for repetition. The same constructs can still be used in EBNF.

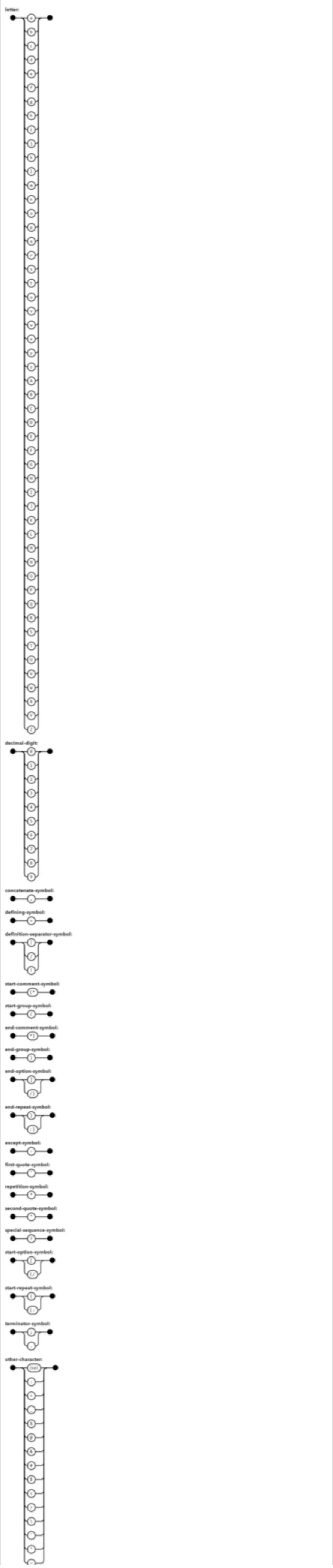
The BNF uses the symbols (<, >, |, ::=) for itself, but does not include quotes around terminal strings. This prevents these characters from being used in the languages, and requires a special symbol for the empty string. In EBNF, terminals are strictly enclosed within quotation marks ("…" or '…'). The angle brackets ("<…>") for nonterminals can be omitted.

BNF syntax can only represent a rule in one line, whereas in EBNF a terminating character, the semicolon character “;” marks the end of a rule.

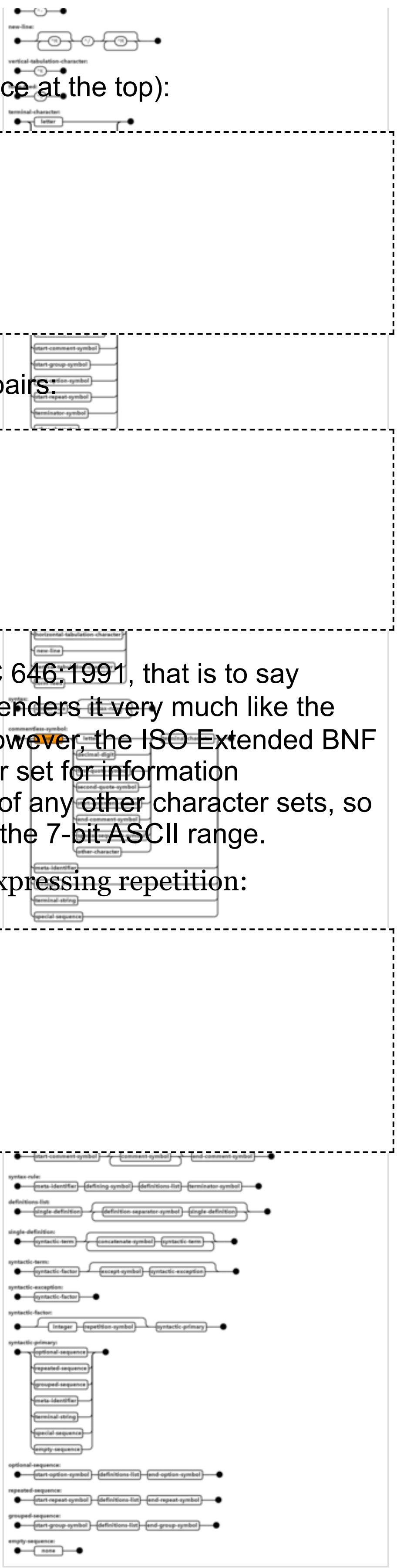
Furthermore, EBNF includes mechanisms for enhancements, defining the number of repetitions, excluding alternatives, comments, etc.

Conventions

1. According to the section 4 of the ISO/IEC 14977 standard, the following conventions are used:
   - Each meta-identifier of Extended BNF is written as one or more words joined together by hyphens. However, joining the words seems to apply only to referencing meta-identifiers outside of the metalanguage itself, as seen in the examples of the standard.



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



One possible EBNF syntax diagram

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
. terminator-symbol
```

3. The normal precedence is overridden by the following bracket pairs:

```
(* start-comment-symbol          end-comment-symbol *)
'  first-quote-symbol            first-quote-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  ?
"  second-quote-symbol          second-quote-symbol  "
```

The first-quote-symbol is the apostrophe as defined by ISO/IEC 646:1991, that is to say Unicode U+0027 ('); the font used in ISO/IEC 14977:1996(E) renders it very much like the acute, Unicode U+00B4 (´), so confusion sometimes arises. However, the ISO Extended BNF standard invokes ISO/IEC 646:1991, "ISO 7-bit coded character set for information interchange", as a normative reference and makes no mention of any other character sets, so formally, there is no confusion with Unicode characters outside the 7-bit ASCII range.

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}, "G";
hh = (aa | bb | cc), "H";
```

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
gg: G AAAG AAAAAAG etc.
hh: AH AAABH CH ACH AACH AAACH
```

Extensibility

According to the ISO 14977 standard EBNF is meant to be extensible, and two facilities are mentioned. The first is part of EBNF grammar, the special sequence, which is arbitrary text enclosed with question marks. The interpretation of the text inside a special sequence is beyond the scope of the EBNF standard. For example, the space character could be defined by the following rule:

```
space = ? ASCII character 32 ?;
```

The second facility for extension is using the fact that parentheses in EBNF cannot be placed next to identifiers (they must be concatenated with them). The following is valid EBNF:

```
something = foo, ( bar );
```

The following is *not* valid EBNF:

```
something = foo ( bar );
```

Therefore, an extension of EBNF could use that notation. For example, in a Lisp grammar, function application could be defined by the following rule:

```
function application = list( symbol, { expression } );
```

Related work

- The W3C publishes an EBNF notation (https://www.w3.org/Notation.html).
- The W3C used a different EBNF (http://www.w3.org/TR/REC-xml/#sec-notation) to specify the XML syntax.
- The British Standards Institution published a standard for an EBNF: BS 6154 in 1981.
- The IETF uses augmented BNF (ABNF), specified in RFC 5234 (https://datatracker.ietf.org/doc/html/rfc5234).

See also

- Meta-II – An early compiler writing tool and notation
- Phrase structure rules – The direct equivalent of EBNF in natural languages
- Regular expression
- Spirit Parser Framework

References

1. Roger S. Scowen: Extended BNF — A generic base standard. Software Engineering Standards Symposium 1993.
2. International standard (ISO 14977 (http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=26153)), which is one of many formats for EBNF, is now freely available as Zip-compressed PDF file (http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996(E).zip).
3. Zaytsev, Vadim (March 26–30, 2012). "BNF Was Here: What Have We Done About the Unnecessary Diversity of Notation for Syntactic Definitions?" (https://www.grammarware.net/text/2012/bnf-was-here.pdf) (PDF). *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC '12)*. Riva del Garda, Italy. p. 1.

4. Wheeler, David A. (2019). "Don't Use ISO/IEC 14977 Extended Backus-Naur Form (EBNF)" (https://dwheeler.com/essays/dont-use-iso-14977-ebnf.html). Retrieved 2021-02-26.
5. Pattis, Richard E. "EBNF: A Notation to Describe Syntax" (http://www.ics.uci.edu/~pattis/misc/ebnf2.pdf) (PDF). *ICS.UCI.edu*. University of California, Irvine. p. 1. Retrieved 2021-02-26.

External links

- ISO/IEC 14977 : 1996(E) (http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf)
- BNF/EBNF variants (http://www.cs.man.ac.uk/~pjj/bnf/ebnf.html) – A table by Pete Jinks comparing several syntaxes

Retrieved from "https://en.wikipedia.org/w/index.php?title=Extended_Backus–Naur_form&oldid=1260951905"