WIKIPEDIA

Backus–Naur form

In computer science, **Backus–Naur form** or **Backus normal form** (**BNF**) is a metasyntax notation for context-free grammars, often used to describe the syntax of languages used in computing, such as computer programming languages, document formats, instruction sets and communication protocols. They are applied wherever exact descriptions of languages are needed: for instance, in official language specifications, in manuals, and in textbooks on programming language theory.

Many extensions and variants of the original Backus–Naur notation are used; some are exactly defined, including extended Backus–Naur form (EBNF) and augmented Backus–Naur form (ABNF).

Contents

History

The idea of describing the structure of language using rewriting rules can be traced back to at least the work of Pāṇini, an ancient Indian Sanskrit grammarian and a revered scholar in Hinduism who lived sometime between the 6th and 4th century BCE.[1][2] His notation to describe Sanskrit

word structure is equivalent in power to that of Backus and has many similar properties.

In Western society, grammar was long regarded as a subject for teaching, rather than scientific study; descriptions were informal and targeted at practical usage. In the first half of the 20th century, linguists such as Leonard Bloomfield and Zellig Harris started attempts to formalize the description of language, including phrase structure.

Meanwhile, string rewriting rules as formal logical systems were introduced and studied by mathematicians such as Axel Thue (in 1914), Emil Post (1920s–40s) and Alan Turing (1936). Noam Chomsky, teaching linguistics to students of information theory at MIT, combined linguistics and mathematics by taking what is essentially Thue's formalism as the basis for the description of the syntax of natural language. He also introduced a clear distinction between generative rules (those of context-free grammars) and transformation rules (1956).[3][4]

John Backus, a programming language designer at IBM, proposed a metalanguage of "metalinguistic formulas"[5][6][7] to describe the syntax of the new programming language IAL, known today as ALGOL 58 (1959). His notation was first used in the ALGOL 60 report.

BNF is a notation for Chomsky's context-free grammars. Backus was familiar with Chomsky's work.[8]

As proposed by Backus, the formula defined "classes" whose names are enclosed in angle brackets. For example, `<ab>`. Each of these names denotes a class of basic symbols.[5]

Further development of ALGOL led to ALGOL 60. In the committee's 1963 report, Peter Naur called Backus's notation *Backus normal form*. Donald Knuth argued that BNF should rather be read as *Backus–Naur form*, as it is "not a normal form in the conventional sense",[9] unlike, for instance, Chomsky normal form. The name *Pāṇini Backus form* was also once suggested in view of the fact that the expansion *Backus normal form* may not be accurate, and that Pāṇini had independently developed a similar notation earlier.[10]

BNF is described by Peter Naur in the ALGOL 60 report as *metalinguistic formula*:[11]

> Sequences of characters enclosed in the brackets <> represent metalinguistic variables whose values are sequences of symbols. The marks "::=" and "|" (the latter with the meaning of "or") are metalinguistic connectives. Any mark in a formula, which is not a variable or a connective, denotes itself. Juxtaposition of marks or variables in a formula signifies juxtaposition of the sequence denoted.

Another example from the ALGOL 60 report illustrates a major difference between the BNF metalanguage and a Chomsky context-free grammar. Metalinguistic variables do not require a rule defining their formation. Their formation may simply be described in natural language within the <> brackets. The following ALGOL 60 report section 2.3 comments specification, exemplifies how this works:

For the purpose of including text among the symbols of a program the following "comment" conventions hold:

The sequence of basic symbols:	is equivalent to
; comment <any sequence not containing ';'>;	**;**
begin comment <any sequence not containing ';'>;	**begin**
end <any sequence not containing 'end' or ';' or 'else'>	**end**

Equivalence here means that any of the three structures shown in the left column may be replaced, in any occurrence outside of strings, by the symbol shown in the same line in the right column without any effect on the action of the program.

Naur changed two of Backus's symbols to commonly available characters. The ::= symbol was originally a :≡. The | symbol was originally the word "or" (with a bar over it).[6]:14 Working for IBM, Backus would have had a non-disclosure agreement and could not have talked about his source if it came from an IBM proprietary project.

BNF is very similar to canonical-form boolean algebra equations that are, and were at the time, used in logic-circuit design. Backus was a mathematician and the designer of the FORTRAN programming language. Studies of boolean algebra is commonly part of a mathematics. What we do know is that neither Backus nor Naur described the names enclosed in < > as non-terminals. Chomsky's terminology was not originally used in describing BNF. Naur later described them as classes in ALGOL course materials.[5] In the ALGOL 60 report they were called metalinguistic variables. Anything other than the metasymbols ::=, |, and class names enclosed in < > are symbols of the language being defined. The metasymbols ::= is to be interpreted as "is defined as". The | is used to separate alternative definitions and is interpreted as "or". The metasymbols < > are delimiters enclosing a class name. BNF is described as a metalanguage for talking about ALGOL by Peter Naur and Saul Rosen.[5]

In 1947 Saul Rosen became involved in the activities of the fledgling Association for Computing Machinery, first on the languages committee that became the IAL group and eventually led to ALGOL. He was the first managing editor of the Communications of the ACM. What we do know is that BNF was first used as a metalanguage to talk about the ALGOL language in the ALGOL 60 report. That is how it is explained in ALGOL programming course material developed by Peter Naur in 1962.[5] Early ALGOL manuals by IBM, Honeywell, Burroughs and Digital Equipment Corporation followed the ALGOL 60 report using it as a metalanguage. Saul Rosen in his book[12] describes BNF as a metalanguage for talking about ALGOL. An example of its use as a metalanguage would be in defining an arithmetic expression:

```
<expr> ::= <term>|<expr><addop><term>
```

The first symbol of an alternative may be the class being defined, the repetition, as explained by Naur, having the function of specifying that the

alternative sequence can recursively begin with a previous alternative and can be repeated any number of times.[5] For example, above <expr> is defined as a <term> followed by any number of <addop> <term>.

In some later metalanguages, such as Schorre's META II, the BNF recursive repeat construct is replaced by a sequence operator and target language symbols defined using quoted strings. The < and > brackets were removed. Parentheses () for mathematical grouping were added. The <expr> rule would appear in META II as

```
EXPR = TERM $('+' TERM .OUT('ADD') | '-' TERM .OUT('SUB'));
```

These changes enabled META II and its derivative programming languages to define and extend their own metalanguage, at the cost of the ability to use a natural language description, metalinguistic variable, language construct description. Many spin-off metalanguages were inspired by BNF. See META II, TREE-META, and Metacompiler.

A BNF class describes a language construct formation, with formation defined as a pattern or the action of forming the pattern. The class name expr is described in a natural language as a <term> followed by a sequence <addop> <term>. A class is an abstraction; we can talk about it independent of its formation. We can talk about term, independent of its definition, as being added or subtracted in expr. We can talk about a term being a specific data type and how an expr is to be evaluated having specific combinations of data types. Or even reordering an expression to group data types and evaluation results of mixed types. The natural-language supplement provided specific details of the language class semantics to be used by a compiler implementation and a programmer writing an ALGOL program. Natural-language description further supplemented the syntax as well. The integer rule is a good example of natural and metalanguage used to describe syntax:

```
<integer> ::= <digit>|<integer><digit>
```

There are no specifics on white space in the above. As far as the rule states, we could have space between the digits. In the natural language we complement the BNF metalanguage by explaining that the digit sequence can have no white space between the digits. English is only one of the possible natural languages. Translations of the ALGOL reports were available in many natural languages.

The origin of BNF is not as important as its impact on programming language development. During the period immediately following the publication of the ALGOL 60 report BNF was the basis of many compiler-compiler systems.

Some, like "A Syntax Directed Compiler for ALGOL 60" developed by Edgar T. Irons and "A Compiler Building System" Developed by Brooker and Morris, directly used BNF. Others, like the Schorre Metacompilers, made it into a programming language with only a few changes. <class name> became symbol identifiers, dropping the enclosing <,> and using quoted strings for symbols of the target language. Arithmetic-like grouping provided a simplification that removed using classes where grouping was its only value. The META II arithmetic expression rule shows grouping use. Output expressions placed in a META II rule are used to output code and labels in an assembly language. Rules in META II are

equivalent to a class definitions in BNF. The Unix utility yacc is based on BNF with code production similar to META II. yacc is most commonly used as a parser generator, and its roots are obviously BNF.

BNF today is one of the oldest computer-related languages still in use.

Introduction

A BNF specification is a set of derivation rules, written as

```
<symbol> ::= __expression__
```

where <symbol>[5] is a *nonterminal*, and the __expression__ consists of one or more sequences of symbols; more sequences are separated by the vertical bar "|", indicating a choice, the whole being a possible substitution for the symbol on the left. Symbols that never appear on a left side are *terminals*. On the other hand, symbols that appear on a left side are *non-terminals* and are always enclosed between the pair <>.[5]

The "::=" means that the symbol on the left must be replaced with the expression on the right.

Example

As an example, consider this possible BNF for a U.S. postal address:

```
<postal-address> ::= <name-part> <street-address> <zip-part>

     <name-part> ::= <personal-part> <last-name> <opt-suffix-part> <EOL> | <personal-part> <name-part>

 <personal-part> ::= <initial> "." | <first-name>

<street-address> ::= <house-num> <street-name> <opt-apt-num> <EOL>

      <zip-part> ::= <town-name> "," <state-code> <ZIP-code> <EOL>

<opt-suffix-part> ::= "Sr." | "Jr." | <roman-numeral> | ""
   <opt-apt-num> ::= <apt-num> | ""
```

This translates into English as:

- A postal address consists of a name-part, followed by a street-address part, followed by a zip-code part.

- A name-part consists of either: a personal-part followed by a last name followed by an optional suffix (Jr., Sr., or dynastic number) and end-of-line, or a personal part followed by a name part (this rule illustrates the use of recursion in BNFs, covering the case of people who use multiple first and middle names and initials).
- A personal-part consists of either a first name or an initial followed by a dot.
- A street address consists of a house number, followed by a street name, followed by an optional apartment specifier, followed by an end-of-line.
- A zip-part consists of a town-name, followed by a comma, followed by a state code, followed by a ZIP-code followed by an end-of-line.
- An opt-suffix-part consists of a suffix, such as "Sr.", "Jr." or a roman-numeral, or an empty string (i.e. nothing).
- An opt-apt-num consists of an apartment number or an empty string (i.e. nothing).

Note that many things (such as the format of a first-name, apartment specifier, ZIP-code, and Roman numeral) are left unspecified here. If necessary, they may be described using additional BNF rules.

Further examples

BNF's syntax itself may be represented with a BNF like the following:

```
 <syntax>         ::= <rule> | <rule> <syntax>
 <rule>           ::= <opt-whitespace> "<" <rule-name> ">" <opt-whitespace> "::=" <opt-whitespace> <expression> <line-end>
 <opt-whitespace> ::= " " <opt-whitespace> | ""
 <expression>     ::= <list> | <list> <opt-whitespace> "|" <opt-whitespace> <expression>
 <line-end>       ::= <opt-whitespace> <EOL> | <line-end> <line-end>
 <list>           ::= <term> | <term> <opt-whitespace> <list>
 <term>           ::= <literal> | "<" <rule-name> ">"
 <literal>        ::= '"' <text1> '"' | "'" <text2> "'"
 <text1>          ::= "" | <character1> <text1>
 <text2>          ::= '' | <character2> <text2>
 <character>      ::= <letter> | <digit> | <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" | "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>         ::=  "|" | " " | "!" | "#" | "$" | "%" | "&" | "(" | ")" | "*" | "+" | "," | "-" | "." | "/" | ":" | ";" | ">" | "=" | "<" | "?" | "@" |
"[" | "\" | "]" | "^" | "_" | "`" | "{" | "}" | "~"
 <character1>     ::= <character> | "'"
 <character2>     ::= <character> | '"'
 <rule-name>      ::= <letter> | <rule-name> <rule-char>
 <rule-char>      ::= <letter> | <digit> | "-"
```

Note that "" is the empty string.

The original BNF did not use quotes as shown in `<literal>` rule. This assumes that no whitespace is necessary for proper interpretation of the rule.

`<EOL>` represents the appropriate line-end specifier (in ASCII, carriage-return, line-feed or both depending on the operating system). `<rule-name>` and `<text>` are to be substituted with a declared rule's name/label or literal text, respectively.

In the U.S. postal address example above, the entire block-quote is a syntax. Each line or unbroken grouping of lines is a rule; for example one rule begins with `<name-part> ::=`. The other part of that rule (aside from a line-end) is an expression, which consists of two lists separated by a pipe `|`. These two lists consists of some terms (three terms and two terms, respectively). Each term in this particular rule is a rule-name.

Variants

There are many variants and extensions of BNF, generally either for the sake of simplicity and succinctness, or to adapt it to a specific application. One common feature of many variants is the use of regular expression repetition operators such as `*` and `+`. The extended Backus–Naur form (EBNF) is a common one.

Another common extension is the use of square brackets around optional items. Although not present in the original ALGOL 60 report (instead introduced a few years later in IBM's PL/I definition), the notation is now universally recognised.

Augmented Backus–Naur form (ABNF) and Routing Backus–Naur form (RBNF)[13] are extensions commonly used to describe Internet Engineering Task Force (IETF) protocols.

Parsing expression grammars build on the BNF and regular expression notations to form an alternative class of formal grammar, which is essentially analytic rather than generative in character.

Many BNF specifications found online today are intended to be human-readable and are non-formal. These often include many of the following syntax rules and extensions:

- Optional items enclosed in square brackets: `[<item-x>]`.
- Items existing 0 or more times are enclosed in curly brackets or suffixed with an asterisk (`*`) such as `<word> ::= <letter> {<letter>}` or `<word> ::= <letter> <letter>*` respectively.
- Items existing 1 or more times are suffixed with an addition (plus) symbol, `+`.
- Terminals may appear in bold rather than italics, and non-terminals in plain text rather than angle brackets.
- Where items are grouped, they are enclosed in simple parentheses.

Software using BNF

- ANTLR, another parser generator written in Java
- Qlik Sense, a BI tool, uses a variant of BNF for scripting
- BNF Converter (BNFC[14]), operating on a variant called "labeled Backus–Naur form" (LBNF). In this variant, each production for a given non-terminal is given a label, which can be used as a constructor of an algebraic data type representing that nonterminal. The converter is capable of producing types and parsers for abstract syntax in several languages, including Haskell and Java.
- Coco/R, compiler generator accepting an attributed grammar in EBNF
- DMS Software Reengineering Toolkit, program analysis and transformation system for arbitrary languages
- GOLD BNF parser
- GNU bison, GNU version of yacc
- RPA BNF parser.[15] Online (PHP) demo parsing: JavaScript, XML
- XACT X4MR System,[16] a rule-based expert system for programming language translation
- XPL Analyzer, a tool which accepts simplified BNF for a language and produces a parser for that language in XPL; it may be integrated into the supplied SKELETON program, with which the language may be debugged[17] (a SHARE contributed program, which was preceded by *A Compiler Generator*, ISBN 978-0-13-155077-3)
- Yacc, parser generator (most commonly used with the Lex preprocessor)
- bnfparser^2,[18] a universal syntax verification utility
- bnf2xml,[19] Markup input with XML tags using advanced BNF matching.
- JavaCC,[20] Java Compiler Compiler tm (JavaCC tm) - The Java Parser Generator.
- Racket's parser tools (https://docs.racket-lang.org/br-parser-tools/index.html), lex and yacc-style Parsing (Beautiful Racket edition)
- Belr (https://github.com/BelledonneCommunications/belr), A parser generator written in C++11. It uses ABNF.

See also

- Compiler Description Language (CDL)
- Syntax diagram – railroad diagram
- Translational Backus–Naur form (TBNF)
- Wirth syntax notation – an alternative to BNF from 1977
- Definite clause grammar – a more expressive alternative to BNF used in Prolog
- Van Wijngaarden grammar – used in preference to BNF to define Algol68

- Meta-II – an early compiler writing tool and notation

References

1. "Panini biography" (http://www-gap.dcs.st-and.ac.uk/~history/Biographies/Panini.html). School of Mathematics and Statistics, University of St Andrews, Scotland. Retrieved 2014-03-22.
2. Ingerman, Peter Zilahy (March 1967). " "Pāṇini-Backus Form" Suggested". *Communications of the ACM*. Association for Computing Machinery. **10** (3): 137. doi:10.1145/363162.363165 (https://doi.org/10.1145%2F363162.363165). S2CID 52817672 (https://api.semanticscholar.org/CorpusID:52817672). Ingerman suggests that the Backus Normal Form be renamed to the Pāṇini-Backus Form, to give due credit to Pāṇini as the earliest independent inventor.
3. Chomsky, Noam (1956). "Three models for the description of language" (https://web.archive.org/web/20100919021754/http://chomsky.info/articles/195609--.pdf) (PDF). *IRE Transactions on Information Theory*. **2** (3): 113–24. doi:10.1109/TIT.1956.1056813 (https://doi.org/10.1109%2FTIT.1956.1056813). Archived from the original (http://www.chomsky.info/articles/195609--.pdf) (PDF) on 2010-09-19.
4. Chomsky, Noam (1957). *Syntactic Structures*. The Hague: Mouton.
5. The meaning of syntactic formula may be further explained by saying that words enclosed in the brackets < >, like <ab>, denote classes whose members are sequences of basic symbols. Class designations of this kind are found in any description of a language. For describing ordinary natural languages designation like word, verb, noun, are used. Peter Naur (1961)."A COURSE ON ALGOL PROGRAMMING" (http://dl.acm.org/citation.cfm?id=1064049&CFID=648579078&CFTOKEN=25521706). p. 5, Note 1. Retrieved 26 March 2015.
6. Backus, J. W. (1959). "The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference" (http://www.softwarepreservation.org/projects/ALGOL/paper/Backus-Syntax_and_Semantics_of_Proposed_IAL.pdf/view). *Proceedings of the International Conference on Information Processing*. UNESCO. pp. 125–132.
7. Farrell, James A. (August 1995). "Compiler Basics: Extended Backus Naur Form" (http://www.cs.man.ac.uk/~pjj/farrell/comp2.html#EBNF). Archived (https://web.archive.org/web/20110605061825/http://www.cs.man.ac.uk/~pjj/farrell/comp2.html#EBNF) from the original on 5 June 2011. Retrieved May 11, 2011.
8. Fulton, III, Scott M. (20 March 2007). "John W. Backus (1924 - 2007)" (http://betanews.com/2007/03/20/john-w-backus-1924-2007). BetaNews. Inc. Retrieved Jun 3, 2014.
9. Knuth, Donald E. (1964). "Backus Normal Form vs. Backus Naur Form". *Communications of the ACM*. **7** (12): 735–736. doi:10.1145/355588.365140 (https://doi.org/10.1145%2F355588.365140). S2CID 47537431 (https://api.semanticscholar.org/CorpusID:47537431).
10. Ingerman, P. Z. (1967). " "Pāṇini Backus Form" suggested". *Communications of the ACM*. **10** (3): 137. doi:10.1145/363162.363165 (https://doi.org/10.1145%2F363162.363165). S2CID 52817672 (https://api.semanticscholar.org/CorpusID:52817672).
11. Revised ALGOL 60 report section. 1.1."ALGOL 60" (http://www.masswerk.at/algol60/report.htm). Retrieved April 18, 2015.
12. Saul Rosen (Jan 1967). *Programming Systems and Languages*. McGraw Hill Computer Science Series. New York/NY: McGraw Hill. ISBN 978-0070537088.
13. RBNF (http://tools.ietf.org/html/rfc5511).

14. "BNFC", *Language technology* (http://bnfc.digitalgrammars.com/), SE: Chalmers
15. "Online demo", *RPatk* (http://www.rpatk.net/web/en/onlinedemo.php)
16. "Tools", *Act world* (https://web.archive.org/web/20130129075050/http://www.actworld.com/tools/), archived from the original (http://www.actworld.com/tools/) on 2013-01-29
17. If the target processor is System/360, or related, even up to z/System, and the target language is similar to PL/I (or, indeed, XPL), then the required code "emitters" may be adapted from XPL's "emitters" for System/360.
18. "BNF parser²", *Source forge* (http://bnfparser2.sourceforge.net/) (project)
19. bnf2xml (http://sourceforge.net/projects/bnf2xml/)
20. "JavaCC" (https://web.archive.org/web/20130608172614/https://javacc.java.net/). Archived from the original (https://javacc.java.net/) on 2013-06-08. Retrieved 2013-09-25.

- This article is based on material taken from the *Free On-line Dictionary of Computing* prior to 1 November 2008 and incorporated under the "relicensing" terms of the GFDL, version 1.3 or later.

External links

- Garshol, Lars Marius, *BNF and EBNF: What are they and how do they work?* (http://www.garshol.priv.no/download/text/bnf.html), NO: Priv.
- RFC 5234 (https://tools.ietf.org/html/rfc5234) — Augmented BNF for Syntax Specifications: ABNF.
- RFC 5511 (https://tools.ietf.org/html/rfc5511) — Routing BNF: A Syntax Used in Various Protocol Specifications.
- ISO/IEC 14977:1996(E) *Information technology – Syntactic metalanguage – Extended BNF*, available from "Publicly available", *Standards* (http://standards.iso.org/ittf/PubliclyAvailableStandards/), ISO or from Kuhn, Marcus, *Iso 14977* (http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf) (PDF), UK: CAM (the latter is missing the cover page, but is otherwise much cleaner)

Language grammars

- Bernhard, *Algol-60 BNF* (https://web.archive.org/web/20060925132043/https://www.lrz-muenchen.de/~bernhard/Algol-BNF.html), DE: LRZ München, archived from the original (http://www.lrz-muenchen.de/~bernhard/Algol-BNF.html) on 2006-09-25, the original BNF.
- "BNF grammars for SQL-92, SQL-99 and SQL-2003", *Savage* (http://savage.net.au/SQL/), AU: Net, freely available BNF grammars for SQL.
- "BNF Web Club", *DB research* (https://web.archive.org/web/20070124000335/http://cui.unige.ch/db-research/Enseignement/analyseinfo/BNFweb.html), CH: Unige, archived from the original (http://cui.unige.ch/db-research/Enseignement/analyseinfo/BNFweb.html) on 2007-01-24, retrieved 2007-01-25, freely available BNF grammars for SQL, Ada, Java.
- "Free Programming Language Grammars for Compiler Construction", *Source code* (http://www.thefreecountry.com/sourcecode/grammars.shtml), The free country, freely available BNF/EBNF grammars for C/C++, Pascal, COBOL, Ada 95, PL/I.

- "BNF files related to the STEP standard", *Exp engine* (https://archive.today/20121225083955/http://exp-engine.svn.sourceforge.net/viewvc/exp-engine/engine/trunk/docs/) (SVN), Source forge, archived from the original (http://exp-engine.svn.sourceforge.net/viewvc/exp-engine/engine/trunk/docs/) on 2012-12-25. Includes parts 11, 14, and 21 of the ISO 10303 (STEP) standard.
