Backus–Naur form

In computer science, **Backus–Naur form** (**BNF**, pronounced /ˌbækəs ˈnaʊər/), also known as **Backus normal form**, is a notation system for defining the syntax of programming languages and other formal languages, developed by John Backus and Peter Naur. It is a metasyntax for context-free grammars, providing a precise way to outline the rules of a language's structure.

It has been widely used in official specifications, manuals, and textbooks on programming language theory, as well as to describe document formats, instruction sets, and communication protocols. Over time, variations such as extended Backus–Naur form (EBNF) and augmented Backus–Naur form (ABNF) have emerged, building on the original framework with added features.

Structure

BNF specifications outline how symbols are combined to form syntactically valid sequences. Each BNF consists of three core components: a set of non-terminal symbols, a set of terminal symbols, and a series of derivation rules.[1] Non-terminal symbols represent categories or variables that can be replaced, while terminal symbols are the fixed, literal elements (such as keywords or punctuation) that appear in the final sequence. Derivation rules provide the instructions for replacing non-terminal symbols with specific combinations of symbols.

A derivation rule is written in the format:

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

where:

- `<symbol>`[2] is a non-terminal symbol, enclosed in angle brackets (<>), identifying the category to be replaced
- `::=` is a metasymbol meaning "is replaced by,"
- `__expression__` is the replacement, consisting of one or more sequences of symbols—either terminal symbols (e.g., literal text like "Sr." or ",") or non-terminal symbols (e.g., `<last-name>`)—with options separated by a vertical bar (|) to indicate alternatives.

For example, in the rule `<opt-suffix-part> ::= "Sr." | "Jr." | ""`, the entire line is the derivation rule, "Sr.", "Jr.", and "" (an empty string) are terminal symbols, and `<opt-suffix-part>` is a non-terminal symbol.

Generating a valid sequence involves starting with a designated start symbol and iteratively applying the derivation rules.[3] This process can extend sequences incrementally. To allow flexibility, some BNF definitions include an optional "delete" symbol (represented as an empty alternative, e.g., `<item> ::= <thing> |`), enabling the removal of certain elements while maintaining syntactic validity.[3]

Example

A practical illustration of BNF is a specification for a simplified 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> ::= <first-name> | <initial> "."

 <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" <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).[4]
- 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 a prefix "Apt" followed by an apartment number, or an empty string (i.e. nothing).

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

History

The concept of using rewriting rules to describe language structure traces back to at least Pāṇini, an ancient Indian Sanskrit grammarian who lived sometime between the 6th and 4th centuries BC.[5] His notation for describing Sanskrit word structure is equivalent in power to that of BNF and exhibits many similar properties.[6]

In Western society, grammar was long regarded as a subject for teaching rather than scientific study; descriptions were informal and targeted at practical usage. This perspective shifted in the first half of the 20th century, when linguists such as Leonard Bloomfield and Zellig Harris began attempts to formalize language description, including phrase structure. Meanwhile, mathematicians explored related ideas through string rewriting rules as formal logical systems, such as Axel Thue in 1914, Emil Post in the

1920s–40s,[7] and Alan Turing in 1936. Noam Chomsky, teaching linguistics to students of information theory at MIT combined linguistics and mathematics, adapting Thue's formalism to describe natural language syntax. In 1956, he introduced a clear distinction between generative rules (those of context-free grammars) and transformation rules.[8][9]

BNF itself emerged when John Backus, a programming language designer at IBM, proposed a metalanguage of *metalinguistic formulas* to define the syntax of the new programming language IAL, known today as ALGOL 58, in 1959.[10] This notation was formalized in the ALGOL 60 report, where Peter Naur named it *Backus normal form* in the committee's 1963 report.[11] Whether Backus was directly influenced by Chomsky's work is uncertain.[12][13]

Donald Knuth argued in 1964 that BNF should be read as *Backus–Naur form*, as it is "not a normal form in the conventional sense," unlike Chomsky normal form.[14] In 1967, Peter Zilahy Ingerman suggested renaming it *Pāṇini Backus form* to acknowledge Pāṇini's earlier, independent development of a similar notation.[15]

In the ALGOL 60 report, Naur described BNF as a *metalinguistic formula*:[16]

> 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.

This is exemplified in the report's section 2.3, where comments are specified:

> 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 altered Backus's original symbols for ALGOL 60, changing ⁝≡ to ⁝:= and the overbarred "$\overline{\text{or}}$" to |, using commonly available characters.[17]:14

BNF is very similar to canonical-form Boolean algebra equations (used in logic-circuit design), reflecting Backus's mathematical background as a FORTRAN designer.[18] Studies of Boolean algebra were commonly part of a mathematics curriculum, which may have informed Backus's approach. Neither Backus nor Naur described the names enclosed in < > as non-terminals—Chomsky's terminology was

not originally used in describing BNF. Naur later called them "classes" in 1961 course materials.[18] In the ALGOL 60 report, they were "metalinguistic variables," with other symbols defining the target language.

Saul Rosen, involved with the Association for Computing Machinery since 1947, contributed to the transition from IAL to ALGOL and edited Communications of the ACM. He described BNF as a metalanguage for ALGOL in his 1967 book.[19] Early ALGOL manuals from IBM, Honeywell, Burroughs, and Digital Equipment Corporation followed this usage.

Impact

BNF significantly influenced programming language development, notably as the basis for early compiler-compiler systems. Examples include Edgar T. Irons' "A Syntax Directed Compiler for ALGOL 60" and Brooker and Morris' "A Compiler Building System," which directly utilized BNF.[20] Others, like Schorre's META II, adapted BNF into a programming language, replacing < > with quoted strings and adding operators like $ for repetition, as in:

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

This influenced tools like yacc, a widely used parser generator rooted in BNF principles, and Unix utilities like yacc.[21] BNF remains one of the oldest computer-related notations still referenced today, though its variants often dominate modern applications. Examples of its use as a metalanguage include defining arithmetic expressions:

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

Here, `<expr>` can recursively include itself, allowing repeated additions.

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

BNF representation of itself

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 vertical bar `|`. These two lists consists of some terms (three terms and two terms, respectively). Each term in this particular rule is a rule-name.

Variants

EBNF

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.

ABNF

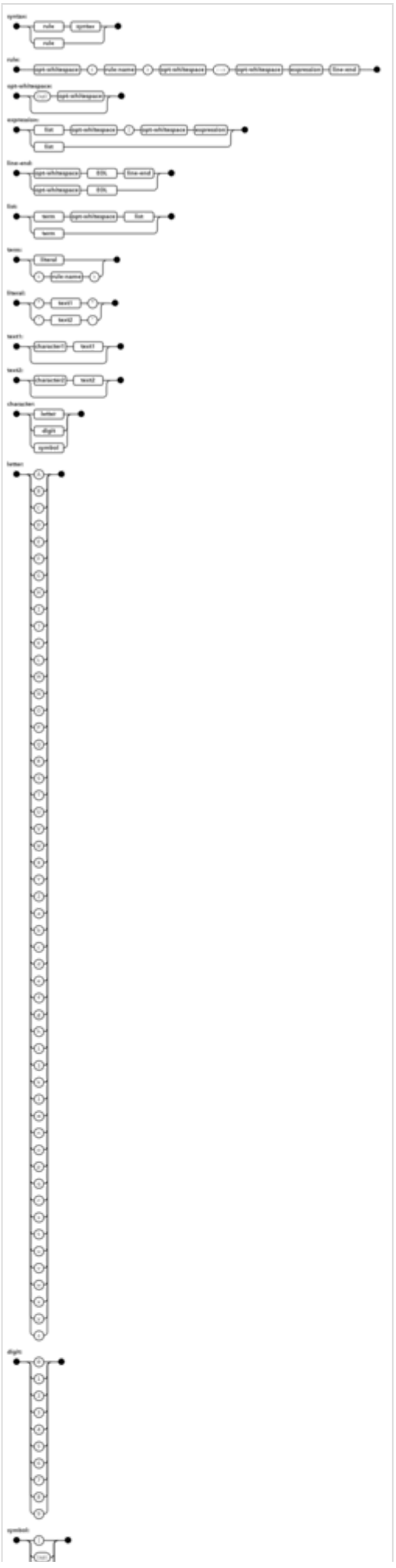
Augmented Backus–Naur form (ABNF) and Routing Backus–Naur form (RBNF)[22] are extensions commonly used to describe Internet Engineering Task Force (IETF) protocols.

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.

Others

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, +, such as `<word> ::= <letter>+`.
- 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.



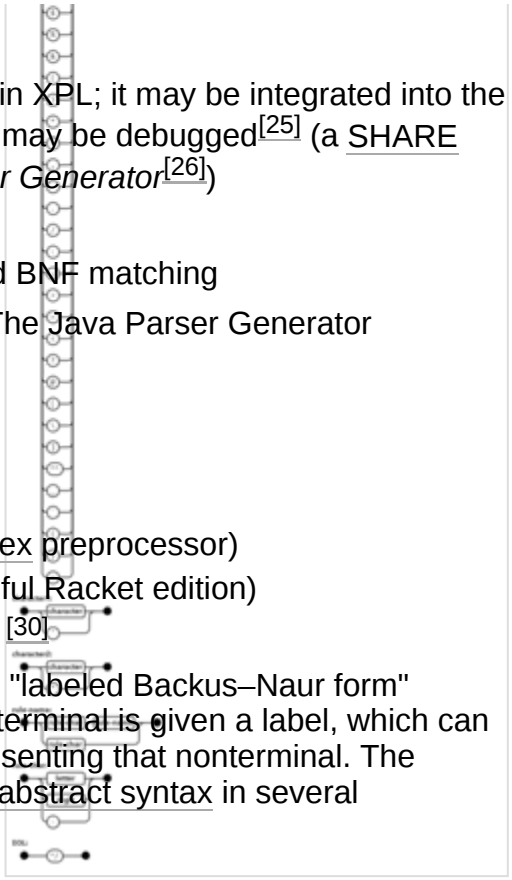
Software using BNF or variants

Software that accepts BNF (or a superset) as input

- ANTLR, a parser generator written in Java
- Coco/R, compiler generator accepting an attributed grammar in EBNF
- DMS Software Reengineering Toolkit, program analysis and transformation system for arbitrary languages
- GOLD, a BNF parser generator
- RPA BNF parser.[23] Online (PHP) demo parsing: JavaScript, XML

- XACT X4MR System,[24] 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[25] (a SHARE contributed program, which was preceded by *A Compiler Generator*[26])
- bnfparser2,[27] a universal syntax verification utility
- bnf2xml,[28] Markup input with XML tags using advanced BNF matching
- JavaCC,[29] Java Compiler Compiler tm (JavaCC tm) - The Java Parser Generator

BNF syntax diagram



Similar software

- GNU bison, GNU version of yacc
- Yacc, parser generator (most commonly used with the Lex preprocessor)
- Racket's parser tools, lex and yacc-style parsing (Beautiful Racket edition)
- Qlik Sense, a BI tool, uses a variant of BNF for scripting [30]
- BNF Converter (BNFC[31]), 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

See also

- Augmented Backus–Naur form (ABNF)
- Compiler Description Language (CDL)
- Definite clause grammar – a more expressive alternative to BNF used in Prolog
- Extended Backus–Naur form (EBNF)
- Meta-II – an early compiler writing tool and notation
- Syntax diagram – railroad diagram
- Translational Backus–Naur form (TBNF)
- Van Wijngaarden grammar – used in preference to BNF to define Algol68
- Wirth syntax notation – an alternative to BNF from 1977

References

1. Janikow, Cezary Z. "What is BNF?" (http://www.cs.umsl.edu/~janikow/cs4280/bnf.pdf) (PDF).
2. Naur, Peter (1961). "A COURSE OF ALGOL 60 PROGRAMMING with special reference to the DASK ALGOL system" (http://archive.computerhistory.org/resources/text/algol/ACM_Algol_bulletin/1064048/frontmatter.pdf) (PDF). Copenhagen: Regnecentralen. Retrieved 26 March 2015.
3. Janikow, Cezary Z. "What is BNF?" (http://www.cs.umsl.edu/~janikow/cs4280/bnf.pdf) (PDF).
4. This article is based on material taken from Backus-Naur+Form (https://foldoc.org/Backus-Naur+Form) at 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.

5. "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.
6. Ingerman, Peter Zilahy (March 1967). " "Pāṇini-Backus Form" Suggested" (https://doi.org/10.1145%2F363162.363165). *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).
7. Post, Emil L. (1943). "Formal Reductions of the General Combinatorial Decision Problem". *American Journal of Mathematics*. **65** (2): 197–215. doi:10.2307/2371804 (https://doi.org/10.2307%2F2371804).
8. Chomsky, Noam (1956). "Three models for the description of language". *IRE Transactions on Information Theory*. **2** (3): 113–24. doi:10.1109/TIT.1956.1056813 (https://doi.org/10.1109%2FTIT.1956.1056813). S2CID 19519474 (https://api.semanticscholar.org/CorpusID:19519474).
9. Chomsky, Noam (1957). *Syntactic Structures*. The Hague: Mouton.
10. Backus, J. W. (1959). "The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference". *Proceedings of the International Conference on Information Processing*. UNESCO. pp. 125–132.
11. Revised ALGOL 60 report section 1.1. "ALGOL 60" (http://www.masswerk.at/algol60/report.htm). Retrieved April 18, 2015.
12. Fulton, Scott M., III (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.
13. John Backus (Sep 2006). Grady Booch (ed.). Oral History of John Backus (https://archive.computerhistory.org/resources/text/Oral_History/Backus_John/Backus_John_1.oral_history.2006.102657970.pdf) (PDF) (Report). Computer History Museum. Here: p.25
14. Knuth, Donald E. (1964). "Backus Normal Form vs. Backus Naur Form" (https://doi.org/10.1145%2F355588.365140). *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).
15. Ingerman, P. Z. (1967). " "Pāṇini Backus Form" suggested" (https://doi.org/10.1145%2F363162.363165). *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).
16. Revised ALGOL 60 report section. 1.1."ALGOL 60" (http://www.masswerk.at/algol60/report.htm). Retrieved April 18, 2015.
17. 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.
18. Naur, Peter (1961). "A COURSE OF ALGOL 60 PROGRAMMING with special reference to the DASK ALGOL system" (http://archive.computerhistory.org/resources/text/algol/ACM_Algol_bulletin/1064048/frontmatter.pdf) (PDF). Copenhagen: Regnecentralen. Retrieved 26 March 2015.
19. Saul Rosen (Jan 1967). *Programming Systems and Languages* (https://archive.org/details/programmingsyste0000unse/mode/1up). McGraw Hill Computer Science Series. New York: McGraw Hill. ISBN 978-0070537088.
20. McKeeman, W. M.; Horning, J.J.; Wortman, D. B. (1970). *A Compiler Generator*. Prentice-Hall. ISBN 978-0-13-155077-3.
21. "BNF parser²", *Source forge* (http://bnfparser2.sourceforge.net/) (project)
22. RBNF (http://tools.ietf.org/html/rfc5511).

23. "Online demo", *RPatk* (https://web.archive.org/web/20121102161746/http://www.rpatk.net/web/en/onlinedemo.php), archived from the original (http://www.rpatk.net/web/en/onlinedemo.php) on 2012-11-02, retrieved 2011-07-03
24. "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
25. 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.
26. McKeeman, W. M.; Horning, J.J.; Wortman, D. B. (1970). *A Compiler Generator* (https://archive.org/details/compilergenerato00mcke). Prentice-Hall. ISBN 978-0-13-155077-3.
27. "BNF parser²", *Source forge* (http://bnfparser2.sourceforge.net/) (project)
28. bnf2xml (http://sourceforge.net/projects/bnf2xml/)
29. "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.
30. "Script Syntax - Qlik Sense on Windows" (https://help.qlik.com/en-US/sense/May2021/Subsystems/Hub/Content/Sense_Hub/Scripting/script-syntax.htm). *Qlik.com*. QlikTech International AB. Retrieved 10 January 2022.
31. "BNFC", *Language technology* (http://bnfc.digitalgrammars.com/), SE: Chalmers

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://www.rfc-editor.org/rfc/rfc5234) — Augmented BNF for Syntax Specifications: ABNF.
- RFC 5511 (https://www.rfc-editor.org/rfc/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* (http://blackbox.userweb.mwn.de/Algol-BNF.html), DE: LRZ München, 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-e

ngine/engine/trunk/docs/) on 2012-12-25. Includes parts 11, 14, and 21 of the ISO 10303 (STEP) standard.
