WIKIPEDIA

# Chomsky normal form

In formal language theory, a context-free grammar, $G$, is said to be in **Chomsky normal form** (first described by Noam Chomsky)[1] if all of its production rules are of the form:[2]

$A \to BC$, or
$A \to a$, or
$S \to \varepsilon$,

where $A$, $B$, and $C$ are nonterminal symbols, the letter $a$ is a terminal symbol (a symbol that represents a constant value), $S$ is the start symbol, and ε denotes the empty string. Also, neither $B$ nor $C$ may be the start symbol, and the third production rule can only appear if ε is in $L(G)$, the language produced by the context-free grammar $G$.[3]:92–93,106

Every grammar in Chomsky normal form is context-free, and conversely, every context-free grammar can be transformed into an equivalent one[note 1] which is in Chomsky normal form and has a size no larger than the square of the original grammar's size.

## Contents



## Converting a grammar to Chomsky normal form

To convert a grammar to Chomsky normal form, a sequence of simple transformations is applied in a certain order; this is described in most textbooks on automata theory.[3]:87–94[4][5][6] The presentation here follows Hopcroft, Ullman (1979), but is adapted to use the transformation names from Lange, Leiß (2009).[7][note 2] Each of the following transformations establishes one of the properties required for Chomsky normal form.

### START: Eliminate the start symbol from right-hand sides

Introduce a new start symbol $S_0$, and a new rule

$$S_0 \to S,$$

where $S$ is the previous start symbol. This does not change the grammar's produced language, and $S_0$ will not occur on any rule's right-hand side.

## TERM: Eliminate rules with nonsolitary terminals

To eliminate each rule

$$A \to X_1 \ldots a \ldots X_n$$

with a terminal symbol $a$ being not the only symbol on the right-hand side, introduce, for every such terminal, a new nonterminal symbol $N_a$, and a new rule

$$N_a \to a.$$

Change every rule

$$A \to X_1 \ldots a \ldots X_n$$

to

$$A \to X_1 \ldots N_a \ldots X_n.$$

If several terminal symbols occur on the right-hand side, simultaneously replace each of them by its associated nonterminal symbol. This does not change the grammar's produced language.[3]:92

## BIN: Eliminate right-hand sides with more than 2 nonterminals

Replace each rule

$$A \to X_1 X_2 \ldots X_n$$

with more than 2 nonterminals $X_1,\ldots,X_n$ by rules

$$A \to X_1 A_1,$$
$$A_1 \to X_2 A_2,$$
$$\ldots,$$
$$A_{n-2} \to X_{n-1} X_n,$$

where $A_i$ are new nonterminal symbols. Again, this does not change the grammar's produced language.[3]:93

## DEL: Eliminate ε-rules

An ε-rule is a rule of the form

$$A \to \varepsilon,$$

where $A$ is not $S_0$, the grammar's start symbol.

To eliminate all rules of this form, first determine the set of all nonterminals that derive ε. Hopcroft and Ullman (1979) call such nonterminals *nullable*, and compute them as follows:

- If a rule $A \to \varepsilon$ exists, then $A$ is nullable.

- If a rule $A \to X_1 \ldots X_n$ exists, and every single $X_i$ is nullable, then $A$ is nullable, too.

Obtain an intermediate grammar by replacing each rule

$$A \to X_1 \ldots X_n$$

by all versions with some nullable $X_i$ omitted. By deleting in this grammar each ε-rule, unless its left-hand side is the start symbol, the transformed grammar is obtained.[3]:90

For example, in the following grammar, with start symbol $S_0$,

$$\begin{aligned}
&S_0 \to AbB \mid C \\
&B \to AA \mid AC \\
&C \to b \mid c \\
&A \to a \mid \varepsilon
\end{aligned}$$

the nonterminal $A$, and hence also $B$, is nullable, while neither $C$ nor $S_0$ is. Hence the following intermediate grammar is obtained:[note 3]

$$\begin{aligned}
&S_0 \to AbB \mid Ab\cancel{B} \mid \cancel{A}bB \mid \cancel{A}b\cancel{B} \;\mid\; C \\
&B \to AA \mid \cancel{A}A \mid A\cancel{A} \mid \cancel{A}\varepsilon\cancel{A} \;\mid\; AC \mid \cancel{A}C \\
&C \to b \mid c \\
&A \to a \mid \varepsilon
\end{aligned}$$

In this grammar, all ε-rules have been "inlined at the call site".[note 4] In the next step, they can hence be deleted, yielding the grammar:

$$\begin{aligned}
&S_0 \to AbB \mid Ab \mid bB \mid b \;\mid\; C \\
&B \to AA \mid A \;\mid\; AC \mid C \\
&C \to b \mid c \\
&A \to a
\end{aligned}$$

This grammar produces the same language as the original example grammar, viz. $\{ab, aba, abaa, abab, abac, abb, abc, b, bab, bac, bb, bc, c\}$, but has no ε-rules.

## UNIT: Eliminate unit rules

A unit rule is a rule of the form

$$A \to B,$$

where $A$, $B$ are nonterminal symbols. To remove it, for each rule

$$B \to X_1 \ldots X_n,$$

where $X_1 \ldots X_n$ is a string of nonterminals and terminals, add rule

$$A \to X_1 \ldots X_n$$

unless this is a unit rule which has already been (or is being) removed.

## Order of transformations

When choosing the order in which the above transformations are to be applied, it has to be considered that some transformations may destroy the result achieved by other ones. For example, **START** will re-introduce a unit rule if it is applied after **UNIT**. The table shows which orderings are admitted.

Moreover, the worst-case bloat in grammar size[note 5] depends on the transformation order. Using $|G|$ to denote the size of the original grammar $G$, the size blow-up in the worst case may range from $|G|^2$ to $2^{2\,|G|}$, depending on the transformation algorithm used.[7]:7 The blow-up in grammar size depends on the order between **DEL** and **BIN**. It may be exponential when **DEL** is done first, but is linear otherwise. **UNIT** can incur a quadratic blow-up in the size of the grammar.[7]:5 The orderings **START**,**TERM**,**BIN**,**DEL**,**UNIT** and **START**,**BIN**,**DEL**,**UNIT**,**TERM** lead to the least (i.e. quadratic) blow-up.

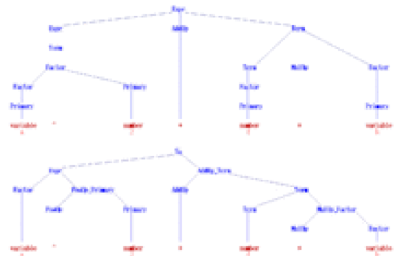
Mutual preservation of transformation results

Transformation *X* always preserves (✓) resp. may destroy (✗) the result of *Y*:

| *X* \ *Y* | START | TERM | BIN | DEL | UNIT |
|---|---|---|---|---|---|
| **START** | | ✓ | ✓ | ✗ | ✗ |
| **TERM** | ✓ | | ✗ | ✓ | ✓ |
| **BIN** | ✓ | ✓ | | ✓ | ✓ |
| **DEL** | ✓ | ✓ | ✓ | | ✗ |
| **UNIT** | ✓ | ✓ | ✓ | (✓)* | |

*__UNIT__ preserves the result of **DEL** if **START** had been called before.

## Example



Abstract syntax tree of the arithmetic expression "*a*^2+4**b*" wrt. the example grammar (**top**) and its Chomsky normal form (**bottom**)

The following grammar, with start symbol *Expr*, describes a simplified version of the set of all syntactical valid arithmetic expressions in programming languages like C or Algol60. Both *number* and *variable* are considered terminal symbols here for simplicity, since in a compiler front-end their internal structure is usually not considered by the parser. The terminal symbol "^" denoted exponentiation in Algol60.

| | | | | |
|---|---|---|---|---|
| *Expr* | → *Term* | \| *Expr AddOp Term* | \| *AddOp Term* |
| *Term* | → *Factor* | \| *Term MulOp Factor* | |
| *Factor* | → *Primary* | \| *Factor* ^ *Primary* | |
| *Primary* | → *number* | \| *variable* | \| ( *Expr* ) |
| *AddOp* | → + | \| − | |
| *MulOp* | → * | \| / | |

In step "START" of the above conversion algorithm, just a rule $S_0 \to Expr$ is added to the grammar. After step "TERM", the grammar looks like this:

| | | | |
|---|---|---|---|
| $S_0$ | → *Expr* | | |
| *Expr* | → *Term* | \| *Expr AddOp Term* | \| *AddOp Term* |
| *Term* | → *Factor* | \| *Term MulOp Factor* | |
| *Factor* | → *Primary* | \| *Factor PowOp Primary* | |
| *Primary* | → *number* | \| *variable* | \| *Open Expr Close* |
| *AddOp* | → + | \| − | |
| *MulOp* | → * | \| / | |
| *PowOp* | → ^ | | |
| *Open* | → ( | | |
| *Close* | → ) | | |

After step "BIN", the following grammar is obtained:

| | | | |
|---|---|---|---|
| $S_0$ | → *Expr* | | |
| *Expr* | → *Term* | \| *Expr AddOp_Term* | \| *AddOp Term* |
| *Term* | → *Factor* | \| *Term MulOp_Factor* | |
| *Factor* | → *Primary* | \| *Factor PowOp_Primary* | |
| *Primary* | → *number* | \| *variable* | \| *Open Expr_Close* |
| *AddOp* | → + | \| − | |
| *MulOp* | → * | \| / | |
| *PowOp* | → ^ | | |
| *Open* | → ( | | |

*Close* → )
*AddOp_Term* → *AddOp Term*
*MulOp_Factor* → *MulOp Factor*
*PowOp_Primary* → *PowOp Primary*
*Expr_Close* → *Expr Close*

Since there are no ε-rules, step "DEL" does not change the grammar. After step "UNIT", the following grammar is obtained, which is in Chomsky normal form:

$S_0$ → *number* | *variable* | *Open Expr_Close* | *Factor PowOp_Primary* | *Term MulOp_Factor* | *Expr AddOp_Term* | *AddOp Term*
*Expr* → *number* | *variable* | *Open Expr_Close* | *Factor PowOp_Primary* | *Term MulOp_Factor* | *Expr AddOp_Term* | *AddOp Term*
*Term* → *number* | *variable* | *Open Expr_Close* | *Factor PowOp_Primary* | *Term MulOp_Factor*
*Factor* → *number* | *variable* | *Open Expr_Close* | *Factor PowOp_Primary*
*Primary* → *number* | *variable* | *Open Expr_Close*
*AddOp* → + | −
*MulOp* → * | /
*PowOp* → ^
*Open* → (
*Close* → )
*AddOp_Term* → *AddOp Term*
*MulOp_Factor* → *MulOp Factor*
*PowOp_Primary* → *PowOp Primary*
*Expr_Close* → *Expr Close*

The $N_a$ introduced in step "TERM" are *PowOp*, *Open*, and *Close*. The $A_i$ introduced in step "BIN" are *AddOp_Term*, *MulOp_Factor*, *PowOp_Primary*, and *Expr_Close*.

# Alternative definition

## Chomsky reduced form

Another way[3]: 92[8] to define the Chomsky normal form is:

A formal grammar is in **Chomsky reduced form** if all of its production rules are of the form:

$A \rightarrow BC$ or
$A \rightarrow a$,

where $A$, $B$ and $C$ are nonterminal symbols, and $a$ is a terminal symbol. When using this definition, $B$ or $C$ may be the start symbol. Only those context-free grammars which do not generate the empty string can be transformed into Chomsky reduced form.

## Floyd normal form

In a letter where he proposed a term Backus–Naur form (BNF), Donald E. Knuth implied a BNF "syntax in which all definitions have such a form may be said to be in 'Floyd Normal Form'",

$\langle A\rangle$ ::= $\langle B\rangle$ | $\langle C\rangle$ or
$\langle A\rangle$ ::= $\langle B\rangle\langle C\rangle$ or
$\langle A\rangle$ ::= $a$,

where $\langle A\rangle$, $\langle B\rangle$ and $\langle C\rangle$ are nonterminal symbols, and $a$ is a terminal symbol, because Robert W. Floyd found any BNF syntax can be converted to the above one in 1961.[9] But he withdrew this term, "since doubtless many people have independently used this simple fact in their own work, and the point is only incidental to the main considerations of Floyd's note."[10] While Floyd's note cites Chomsky's original 1959 article, Knuth's letter does not.

# Application

Besides its theoretical significance, CNF conversion is used in some algorithms as a preprocessing step, e.g., the CYK algorithm, a bottom-up parsing for context-free grammars, and its variant probabilistic CKY.[11]

# See also

- Backus–Naur form
- CYK algorithm
- Greibach normal form
- Kuroda normal form
- Pumping lemma for context-free languages — its proof relies on the Chomsky normal form

# Notes

1. that is, one that produces the same language
2. For example, Hopcroft, Ullman (1979) merged **TERM** and **BIN** into a single transformation.
3. indicating a kept and omitted nonterminal *N* by *N* and ~~N~~, respectively
4. If the grammar had a rule $S_0 \to \varepsilon$, it could not be "inlined", since it had no "call sites". Therefore it could not be deleted in the next step.
5. i.e. written length, measured in symbols

# References


1. Chomsky, Noam (1959). "On Certain Formal Properties of Grammars" (https://doi.org/10.1016%2FS0019-9958%2859%2990362-6). *Information and Control*. **2** (2): 137–167. doi:10.1016/S0019-9958(59)90362-6 (https://doi.org/10.1016%2FS0019-9958%2859%2990362-6). Here: Sect.6, p.152ff.
2. D'Antoni, Loris. "Page 7, Lecture 9: Bottom-up Parsing Algorithms" (http://pages.cs.wisc.edu/~loris/cs536/slides/lec9.pdf) (PDF). *CS536-S21 Intro to Programming Languages and Compilers*. University of Wisconsin-Madison.
3. Hopcroft, John E.; Ullman, Jeffrey D. (1979). *Introduction to Automata Theory, Languages and Computation* (https://archive.org/details/introductiontoau00hopc). Reading, Massachusetts: Addison-Wesley Publishing. ISBN 978-0-201-02988-8.
4. Hopcroft, John E.; Motwani, Rajeev; Ullman, Jeffrey D. (2006). *Introduction to Automata Theory, Languages, and Computation* (https://archive.org/details/introductiontoau0000hopc) (3rd ed.). Addison-Wesley. ISBN 978-0-321-45536-9. Section 7.1.5, p.272
5. Rich, Elaine (2007). *Automata, Computability, and Complexity: Theory and Applications* (1st ed.). Prentice-Hall. ISBN 978-0132288064.
6. Wegener, Ingo (1993). *Theoretische Informatik - Eine algorithmenorientierte Einführung*. Leitfäden und Mongraphien der Informatik (in German). Stuttgart: B. G. Teubner. ISBN 978-3-519-02123-0. Section 6.2 "Die Chomsky-Normalform für kontextfreie Grammatiken", p. 149–152
7. Lange, Martin; Leiß, Hans (2009). "To CNF or not to CNF? An Efficient Yet Presentable Version of the CYK Algorithm" (http://ddi.cs.uni-potsdam.de/InformaticaDidactica/LangeLeiss2009.pdf) (PDF). *Informatica Didactica*. **8**.

8. Hopcroft et al. (2006)
9. Floyd, Robert W. (1961). "Note on mathematical induction in phrase structure grammars" (https://core.ac.uk/download/pdf/82003923.pdf) (PDF). *Information and Control*. **4** (4): 353–358. doi:10.1016/S0019-9958(61)80052-1 (https://doi.org/10.1016%2FS0019-9958%2861%2980052-1). Here: p.354
10. Knuth, Donald E. (December 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).
11. Jurafsky, Daniel; Martin, James H. (2008). *Speech and Language Processing* (2nd ed.). Pearson Prentice Hall. p. 465. ISBN 978-0-13-187321-6.

# Further reading

- Cole, Richard. *Converting CFGs to CNF (Chomsky Normal Form)*, October 17, 2007. (pdf) (http://cs.nyu.edu/courses/fall07/V22.0453-001/cnf.pdf) — uses the order TERM, BIN, START, DEL, UNIT.
- John Martin (2003). *Introduction to Languages and the Theory of Computation*. McGraw Hill. ISBN 978-0-07-232200-2. *(Pages 237–240 of section 6.6: simplified forms and normal forms.)*
- Michael Sipser (1997). *Introduction to the Theory of Computation* (https://archive.org/details/introductiontoth00sips). PWS Publishing. ISBN 978-0-534-94728-6. *(Pages 98–101 of section 2.1: context-free grammars. Page 156.)*
- Sipser, Michael. *Introduction to the Theory of Computation,* 2nd edition.
- Alexander Meduna (6 December 2012). *Automata and Languages: Theory and Applications* (https://books.google.com/books?id=a-rjBwAAQBAJ&q=%22Chomsky+normal+form%22). Springer Science & Business Media. ISBN 978-1-4471-0501-5.