Conjunctive normal form

In Boolean logic, a formula is in **conjunctive normal form (CNF)** or **clausal normal form** if it is a conjunction of one or more clauses, where a clause is a disjunction of literals; otherwise put, it is **an AND of ORs**. As a normal form, it is useful in automated theorem proving. It is similar to the product of sums form used in circuit theory.

All conjunctions of literals and all disjunctions of literals are in CNF, as they can be seen as conjunctions of one-literal clauses and conjunctions of a single clause, respectively. As in the disjunctive normal form (DNF), the only propositional connectives a formula in CNF can contain are and, or, and not. The not operator can only be used as part of a literal, which means that it can only precede a propositional variable or a predicate symbol.

In automated theorem proving, the notion "*clausal normal form*" is often used in a narrower sense, meaning a particular representation of a CNF formula as a set of sets of literals.

Contents

1 Examples and non-examples
2 Conversion into CNF
3 First-order logic
4 Computational complexity
5 Converting from first-order logic
6 Notes
7 See also
8 References
9 External links

Examples and non-examples

All of the following formulas in the variables A, B, C, D, and E are in conjunctive normal form:

- $\neg A \wedge (B \vee C)$
- $(A \vee B) \wedge (\neg B \vee C \vee \neg D) \wedge (D \vee \neg E)$
- $A \vee B$
- $A \wedge B$

The third formula is in conjunctive normal form because it is viewed as a "conjunction" with just one conjunct, namely the clause $A \vee B$. Incidentally, the last two formulas are also in disjunctive normal form.

The following formulas are *not* in conjunctive normal form:

- $\neg(B \vee C)$, since an OR is nested within a NOT
- $(A \wedge B) \vee C$
- $A \wedge (B \vee (D \wedge E))$, since an AND is nested within an OR

Every formula can be equivalently written as a formula in conjunctive normal form. In particular this is the case for the three non-examples just mentioned; they are respectively equivalent to the following three formulas, which are in conjunctive normal form:

- $\neg B \wedge \neg C$
- $(A \vee C) \wedge (B \vee C)$
- $A \wedge (B \vee D) \wedge (B \vee E)$.

Conversion into CNF

Every propositional formula can be converted into an equivalent formula that is in CNF. This transformation is based on rules about logical equivalences: the double negative law, De Morgan's laws, and the distributive law.

Since all logical formulæ can be converted into an equivalent formula in conjunctive normal form, proofs are often based on the assumption that all formulae are CNF. However, in some cases this conversion to CNF can lead to an exponential explosion of the formula. For example, translating the following non-CNF formula into CNF produces a formula with 2^n clauses:

$$(X_1 \wedge Y_1) \vee (X_2 \wedge Y_2) \vee \cdots \vee (X_n \wedge Y_n).$$

In particular, the generated formula is:

$$(X_1 \vee X_2 \vee \cdots \vee X_n) \wedge (Y_1 \vee X_2 \vee \cdots \vee X_n) \wedge (X_1 \vee Y_2 \vee \cdots \vee X_n) \wedge (Y_1 \vee Y_2 \vee \cdots \vee X_n) \wedge \cdots \wedge (Y_1 \vee Y_2 \vee \cdots \vee Y_n).$$

This formula contains 2^n clauses; each clause contains either X_i or Y_i for each i.

There exist transformations into CNF that avoid an exponential increase in size by preserving satisfiability rather than equivalence.[1][2] These transformations are guaranteed to only linearly increase the size of the formula, but introduce new variables. For example, the above formula can be transformed into CNF by adding variables $Z_1, \ldots, Z_n$ as follows:

$$(Z_1 \vee \cdots \vee Z_n) \wedge (\neg Z_1 \vee X_1) \wedge (\neg Z_1 \vee Y_1) \wedge \cdots \wedge (\neg Z_n \vee X_n) \wedge (\neg Z_n \vee Y_n).$$

An interpretation satisfies this formula only if at least one of the new variables is true. If this variable is Z_i, then both X_i and Y_i are true as well. This means that every model that satisfies this formula also satisfies the original one. On the other hand, only some of the models of the original formula satisfy this one: since the Z_i are not mentioned in the original formula, their values are irrelevant to satisfaction of it, which is not the case in the last formula. This means that the original formula and the result of the translation are equisatisfiable but not equivalent.

An alternative translation, the Tseitin transformation, includes also the clauses $Z_i \vee \neg X_i \vee \neg Y_i$. With these clauses, the formula implies $Z_i \equiv X_i \wedge Y_i$; this formula is often regarded to "define" Z_i to be a name for $X_i \wedge Y_i$.

First-order logic

In first order logic, conjunctive normal form can be taken further to yield the clausal normal form of a logical formula, which can be then used to perform first-order resolution. In resolution-based automated theorem-proving, a CNF formula

$(\ l_{11} \vee \ldots \vee l_{1n_1}\ ) \wedge \ldots \wedge (\ l_{m1} \vee \ldots \vee l_{mn_m}\ )$, with l_{ij} literals, is commonly represented as a set of sets $\{\{\ l_{11}\ ,\ \ldots\ ,\ l_{1n_1}\ \}\ ,\ \ldots\ ,\ \{\ l_{m1}\ ,\ \ldots\ ,\ l_{mn_m}\ \}\}$.

See below for an example.

Computational complexity

An important set of problems in computational complexity involves finding assignments to the variables of a boolean formula expressed in Conjunctive Normal Form, such that the formula is true. The k-SAT problem is the problem of finding a satisfying assignment to a boolean formula expressed in CNF in which each disjunction contains at most k variables. 3-SAT is NP-complete (like any other k-SAT problem with k>2) while 2-SAT is known to have solutions in polynomial time. As a consequence,[3] the task of converting a formula into a DNF, preserving satisfiability, is NP-hard; dually, converting into CNF, preserving validity, is also NP-hard; hence equivalence-preserving conversion into DNF or CNF is again NP-hard.

Typical problems in this case involve formulas in "3CNF": conjunctive normal form with no more than three variables per conjunct. Examples of such formulas encountered in practice can be very large, for example with 100,000 variables and 1,000,000 conjuncts.

A formula in CNF can be converted into an equisatisfiable formula in "kCNF" (for k≥3) by replacing each conjunct with more than k variables $X_1 \vee \cdots \vee X_k \vee \cdots \vee X_n$ by two conjuncts $X_1 \vee \cdots \vee X_{k-1} \vee Z$ and $\neg Z \vee X_k \cdots \vee X_n$ with Z a new variable, and repeating as often as necessary.

Converting from first-order logic

To convert first-order logic to CNF:[4]

1. Convert to negation normal form
   1. Eliminate implications and equivalences: repeatedly replace $P \to Q$ with $\neg P \vee Q$; replace $P \leftrightarrow Q$ with $(P \vee \neg Q) \wedge (\neg P \vee Q)$. Eventually, this will eliminate all occurrences of $\to$ and $\leftrightarrow$.
   2. Move NOTs inwards by repeatedly applying De Morgan's Law. Specifically, replace $\neg(P \vee Q)$ with $(\neg P) \wedge (\neg Q)$; replace $\neg(P \wedge Q)$ with $(\neg P) \vee (\neg Q)$; and replace $\neg\neg P$ with P; replace $\neg(\forall x P(x))$ with $\exists x \neg P(x)$; $\neg(\exists x P(x))$ with $\forall x \neg P(x)$. After that, a $\neg$ may occur only immediately before a predicate symbol.
2. Standardize variables
   1. For sentences like $(\forall x P(x)) \vee (\exists x Q(x))$ which use the same variable name twice, change the name of one of the variables. This avoids confusion later when dropping quantifiers. For example, $\forall x [\exists y \mathbf{Animal}(y) \wedge \neg \mathbf{Loves}(x, y)] \vee [\exists y \mathbf{Loves}(y, x)]$ is renamed to $\forall x [\exists y \mathbf{Animal}(y) \wedge \neg \mathbf{Loves}(x, y)] \vee [\exists z \mathbf{Loves}(z, x)]$.
3. Skolemize the statement
   1. Move quantifiers outwards: repeatedly replace $P \wedge (\forall x Q(x))$ with $\forall x (P \wedge Q(x))$; replace $P \vee (\forall x Q(x))$ with $\forall x (P \vee Q(x))$; replace $P \wedge (\exists x Q(x))$ with $\exists x (P \wedge Q(x))$; replace $P \vee (\exists x Q(x))$ with $\exists x (P \vee Q(x))$. These replacements preserve equivalence, since the previous variable standardization step ensured that x doesn't occur in P. After these replacements, a quantifier may occur only in the initial prefix of the formula, but never inside a $\neg$, $\wedge$, or $\vee$.
   2. Repeatedly replace $\forall x_1 \ldots \forall x_n\ \exists y\ P(y)$ with $\forall x_1 \ldots \forall x_n\ P(f(x_1, \ldots, x_n))$, where f is a new n-ary function symbol, a so-called 'skolem function'. This is the only step that preserves only satisfiability rather than equivalence. It eliminates all existential quantifiers.
4. Drop all universal quantifiers.
5. Distribute ORs inwards over ANDs: repeatedly replace $P \vee (Q \wedge R)$ with $(P \vee Q) \wedge (P \vee R)$.

As an example, the formula saying *"Anyone who loves all animals, is in turn loved by someone"* is converted into CNF (and subsequently into clause form in the last line) as follows (highlighting replacement rule redices in **red**):

$$
\begin{array}{lllllllll}
\forall x & (\forall y & & \mathbf{Animal}(y) &) \to & \mathbf{Loves}(x, y) &) & \to (\exists y\, \mathbf{Loves}(y & , x)) \\
\forall x & (\forall y & \neg & \mathbf{Animal}(y) &) \vee & \mathbf{Loves}(x, y) &) & \to (\exists y\, \mathbf{Loves}(y & , x)) \\
\forall x\ \neg & (\forall y & \neg & \mathbf{Animal}(y) &) \vee & \mathbf{Loves}(x, y) &) & \vee (\exists y\, \mathbf{Loves}(y & , x)) \\
\forall x & (\exists y\ \neg (& \neg & \mathbf{Animal}(y) &) \vee & \mathbf{Loves}(x, y) &)) & \vee (\exists y\, \mathbf{Loves}(y & , x)) \\
\forall x & (\exists y & \neg\ \neg & \mathbf{Animal}(y) &) \wedge \neg & \mathbf{Loves}(x, y) &) & \vee (\exists y\, \mathbf{Loves}(y & , x)) \\
\forall x & (\exists y & & \mathbf{Animal}(y) &) \wedge \neg & \mathbf{Loves}(x, y) &) & \vee (\exists y\, \mathbf{Loves}(y & , x)) \\
\forall x & (\exists y & & \mathbf{Animal}(y) &) \wedge \neg & \mathbf{Loves}(x, y) &) & \vee (\exists z\, \mathbf{Loves}(z & , x)) \\
\forall x\ \exists z & (\exists y & & \mathbf{Animal}(y) &) \wedge \neg & \mathbf{Loves}(x, y) &) & \vee \quad \mathbf{Loves}(z & , x) \\
\forall x\ \exists z & \exists y & (& \mathbf{Animal}(y) &) \wedge \neg & \mathbf{Loves}(x, y) &)) & \vee \quad \mathbf{Loves}(z & , x) \\
\forall x & \exists y & (& \mathbf{Animal}(y) &) \wedge \neg & \mathbf{Loves}(x, y) &)) & \vee \quad \mathbf{Loves}(g(x) & , x) \\
 & & (& \mathbf{Animal}(f(x)) &) \wedge \neg & \mathbf{Loves}(x, f(x)) &)) & \vee \quad \mathbf{Loves}(g(x) & , x) \\
 & (& & \mathbf{Animal}(f(x)) & & & & \vee \quad \mathbf{Loves}(g(x) & , x) &) \wedge (\neg \mathbf{Loves}(x, f(x)) \vee \mathbf{Loves}(g(x), x)) \\
 & \{\{ & & \mathbf{Animal}(f(x)) & & & & , \quad \mathbf{Loves}(g(x) & , x) & \}\ ,\ \{\neg \mathbf{Loves}(x, f(x))\ ,\ \mathbf{Loves}(g(x), x)\}
\end{array}
$$

Informally, the skolem function $g(x)$ can be thought of as yielding the person by whom x is loved, while $f(x)$ yields the animal (if any) that x doesn't love. The 3rd last line from below then reads as *"x doesn't love the animal $f(x)$, or else x is loved by $g(x)$"*.

The 2nd last line from above, $(\mathbf{Animal}(f(x)) \vee \mathbf{Loves}(g(x), x)) \wedge (\neg \mathbf{Loves}(x, f(x)) \vee \mathbf{Loves}(g(x), x))$, is the CNF.

Notes

1. Tseitin (1968)

2. Jackson and Sheridan (2004)
3. since one way to check a CNF for satisfiability is to convert it into a DNF, the satisfiability of which can be checked in linear time
4. Artificial Intelligence: A modern Approach (https://pdfs.semanticscholar.org/bef0/731f247a1d01c9e0ff52f2412007c143899d.pdf) [1995...] Russell and Norvig

See also

- Algebraic normal form
- Disjunctive normal form
- Horn clause
- Quine–McCluskey algorithm

References

- Paul Jackson, Daniel Sheridan: Clause Form Conversions for Boolean Circuits. In: Holger H. Hoos, David G. Mitchell (Eds.): Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10–13, 2004, Revised Selected Papers. Lecture Notes in Computer Science 3542, Springer 2005, pp. 183–198
- G.S. Tseitin: On the complexity of derivation in propositional calculus. In: Slisenko, A.O. (ed.) Structures in Constructive Mathematics and Mathematical Logic, Part II, Seminars in Mathematics (translated from Russian), pp. 115–125. Steklov Mathematical Institute (1968)

External links

- Hazewinkel, Michiel, ed. (2001) [1994], "Conjunctive normal form", *Encyclopedia of Mathematics*, Springer Science+Business Media B.V. / Kluwer Academic Publishers, ISBN 978-1-55608-010-4
- Java applet for converting to CNF and DNF showing laws used
- Mayuresh S. Pardeshi, Dr Bashirahamed F. Momin "Conversion of cnf to dnf using grid computing" IEEE, ISBN 978-1-4673-2816-6
- Mayuresh S. Pardeshi, Dr Bashirahamed F. Momin "Conversion of cnf to dnf using grid computing in parallel" IEEE, ISBN 978-1-4799-4041-7