

# Formal Methods in Software Engineering

**Normal Forms** — Spring 2025

Konstantin Chukharev

# §1 Normal Forms

# Normal Forms in Propositional Logic

**Definition 1** (Normal form): A *normal form* is a standardized syntactic representation of logical formulas with a *restricted* structure.

Normal forms enable efficient reasoning, simplification, and decision procedures, making them essential in automated theorem proving, model checking, and logic synthesis.

There are several *normal forms* commonly used in propositional logic:

- Negation normal form (NNF)
- Conjunctive normal form (CNF)
- Disjunctive normal form (DNF)
- Algebraic normal form (ANF)
- Binary decision diagram (BDD)

Each normal form has its own advantages and disadvantages, and is used in different contexts.

Every propositional formula can be converted to an *equivalent* formula in any of these normal forms.

# Negation Normal Form

**Definition 2** (Negation Normal Form (NNF)): A formula is in *negation normal form* if the negation operator ( $\neg$ ) is only applied to variables, and the only allowed logical connectives are  $\wedge$  and  $\vee$ .

**Example:** The formula  $(p \wedge q) \vee (\neg p \wedge \neg q)$  is in NNF.

**Example:** The formula  $\neg(p \wedge q) \vee (\neg p \wedge \neg q)$  is *not* in NNF due to  $\neg(\dots)$ .

**Grammar** for NNF formulas:

$$\langle \text{Atom} \rangle ::= \top \mid \perp \mid \langle \text{Variable} \rangle$$
$$\langle \text{Literal} \rangle ::= \langle \text{Atom} \rangle \mid \neg \langle \text{Atom} \rangle$$
$$\langle \text{Formula} \rangle ::= \langle \text{Literal} \rangle \mid \langle \text{Formula} \rangle \wedge \langle \text{Formula} \rangle \mid \langle \text{Formula} \rangle \vee \langle \text{Formula} \rangle$$

# Literals

**Definition 3** (Literal): A *literal* is a propositional variable or its negation.

- $p$  is a *positive literal*.
- $\neg p$  is a *negative literal*.

**Definition 4** (Complement): The *complement* of a literal  $p$  is denoted by  $\bar{p}$ .

$$\bar{p} = \begin{cases} \neg p & \text{if } p \text{ is positive} \\ p & \text{if } p \text{ is negative} \end{cases}$$

Note: *complementary* literals  $p$  and  $\bar{p}$  are each other's complement.

## NNF Transformation

Any propositional formula can be converted to NNF by the repeated application of the following rewriting rules ( $\implies$ ) to the formula and its sub-formulas, to completion (until none apply):

Description	Rewrite rule
Eliminate implications	$(A \rightarrow B) \implies (\neg A \vee B)$
Eliminate bi-implications	$(A \leftrightarrow B) \implies (\neg A \vee B) \wedge (A \vee \neg B)$
Push negation inside conjunctions	$\neg(A \wedge B) \implies (\neg A \vee \neg B)$
Push negation inside disjunctions	$\neg(A \vee B) \implies \neg A \wedge \neg B$
Eliminate double negations	$\neg\neg A \implies A$

**Theorem 1:** Every well-formed formula not containing  $\leftrightarrow$  can be converted to an *equivalent* NNF with a *linear increase* in the size<sup>1</sup> of the formula.

---

<sup>1</sup>For example, number of variable occurrences, or number of sub-formulas.

## Exponential Blowup of NNF

The NNF of formulas containing  $\leftrightarrow$  can grow *exponentially* in size.

**Example:** Let's convert the following formula to NNF...

$$\begin{aligned} F &= a \leftrightarrow (b \leftrightarrow (c \leftrightarrow d)) \implies \\ &= a \leftrightarrow (b \leftrightarrow ((c \rightarrow d) \wedge (d \rightarrow c))) \implies \\ &= a \leftrightarrow ((b \rightarrow ((c \rightarrow d) \wedge (d \rightarrow c))) \wedge (((c \rightarrow d) \wedge (d \rightarrow c)) \rightarrow b)) \implies \\ &= a \leftrightarrow ((b \vee (\dots)) \wedge (\neg(\dots) \vee b)) \implies \\ &= (\neg a \vee (\dots)) \wedge (a \vee \neg(\dots)) \implies \\ &= (\neg a \vee ((b \vee (\dots)) \wedge (\neg(\dots) \vee b))) \wedge \\ &\quad (a \vee \neg((b \vee (\dots)) \wedge (\neg(\dots) \vee b))) \end{aligned}$$

The original  $F$  contains only 4 variable occurrences, while the NNF of  $F$  contains 16 variable occurrences.

## Disjunctive Normal Form

**Definition 5** (Disjunctive Normal Form (DNF)): A formula is said to be in *disjunctive normal form* if it is a disjunction of *cubes* (conjunctions of literals).

$$A = \bigvee_i \bigwedge_j p_{ij}$$

**Example:**  $A = (p \wedge q) \vee (\neg p \wedge q \wedge r) \vee \neg q$

**Grammar** for DNF formulas:

$\langle \text{Atom} \rangle ::= \top \mid \perp \mid \langle \text{Variable} \rangle$

$\langle \text{Literal} \rangle ::= \langle \text{Atom} \rangle \mid \neg \langle \text{Atom} \rangle$

$\langle \text{Cube} \rangle ::= \langle \text{Literal} \rangle \mid \langle \text{Literal} \rangle \wedge \langle \text{Cube} \rangle$

$\langle \text{Formula} \rangle ::= \langle \text{Cube} \rangle \mid \langle \text{Cube} \rangle \vee \langle \text{Formula} \rangle$



## Cubes and Clauses

**Definition 6** (Cube): A *cube* is a conjunction of literals.

**Definition 7** (Clause): A *clause* is a disjunction of literals.

- An *empty clause* is a clause with no literals, commonly denoted by  $\square$ .
- A *unit clause* is a clause with a single literal, that is, just a literal itself.
- A *Horn clause* is a clause with at most one positive literal.

**Note:**  $\square$  is *false in every interpretation*, that is, unsatisfiable.

## Conjunctive Normal Form

**Definition 8** (Conjunctive Normal Form (CNF)): A formula is said to be in *conjunctive normal form* if it is a conjunction of *clauses*.

$$A = \bigwedge_i \bigvee_j p_{ij}$$

**Example:**  $A = (\neg p \vee q) \wedge (\neg p \vee q \vee r) \wedge \neg q$

## Satisfiability on CNF

An interpretation  $\nu$  satisfies a clause  $C = p_1 \vee \dots \vee p_n$  if it satisfies some (at least one) literal  $p_k$  in  $C$ .

An interpretation  $\nu$  satisfies a CNF formula  $A = C_1 \wedge \dots \wedge C_n$  if it satisfies every clause  $C_i$  in  $A$ .

A CNF formula  $A$  is *satisfiable* if there exists an interpretation  $\nu$  that satisfies  $A$ .

The **SAT problem** is about determining whether a given CNF formula is satisfiable.

## CNF Transformation

Any propositional formula can be converted to CNF by the repeated application of these rewriting rules:

- Any NNF transformation rules.
- Distribute  $\vee$  over  $\wedge$  (another source of exponential blowup):
  - ▶  $A \vee (B \wedge C) \implies (A \vee B) \wedge (A \vee C)$
  - ▶  $(A \wedge B) \vee C \implies (A \vee C) \wedge (B \vee C)$
- Normalize nested  $\wedge$  and  $\vee$  operators:
  - ▶  $A \wedge (B \wedge C) \implies (A \wedge B \wedge C)$
  - ▶  $A \vee (B \vee C) \implies (A \vee B \vee C)$

**Theorem 2:** Every well-formed formula  $\alpha$  can be converted to an *equivalent* CNF  $\alpha'$  with a *potentially exponential increase* in the size of the formula.

## Exponential Blowup of CNF

Distributive law is the main source of the exponential blowup in CNF conversion:

$$n \text{ cubes} \left\{ \begin{array}{l} (x_1 \wedge y_1) \vee \\ (x_2 \wedge y_2) \vee \\ \dots \\ (x_n \wedge y_n) \vee \end{array} \right. \xRightarrow{\text{CNF}} \left\{ \begin{array}{l} (x_1 \vee x_2 \vee \dots \vee x_n) \wedge \\ (y_1 \vee x_2 \vee \dots \vee x_n) \wedge \\ \dots \\ (x_1 \vee y_2 \vee \dots \vee y_n) \wedge \\ (y_1 \vee y_2 \vee \dots \vee y_n) \end{array} \right\} 2^n \text{ clauses}$$

Is there a way to avoid the exponential blowup? *Yes!*

## Tseitin Transformation

A space-efficient way to convert a formula to CNF is the *Tseitin transformation*, which is based on so-called “*naming*” or “*definition introduction*”, allowing to replace subformulas with the “*fresh*” (new) variables.

1. Take a subformula  $A$  of a formula  $F$ .
2. Introduce a new propositional variable  $n$ .
3. Add a *definition* for  $n$ , that is, a formula stating that  $n$  is equivalent to  $A$ .
4. Replace  $A$  with  $n$  in  $F$ .

Overall, construct  $S := F[n/A] \wedge (n \leftrightarrow A)$

$$\begin{aligned} F &= p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow \overbrace{(p_5 \leftrightarrow p_6)}^A))) \implies \\ S &= p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow n))) \wedge \\ &\quad n \leftrightarrow (p_5 \leftrightarrow p_6) \end{aligned}$$

**Note:** The resulting formula is, in general, **not equivalent** to the original one, but it is *equisatisfiable*, i.e., it is satisfiable iff the original formula is satisfiable.

## Equisatisfiability

**Definition 9** (Equisatisfiability): Two formulas  $A$  and  $B$  are *equisatisfiable* if  $A$  is satisfiable *if and only if*  $B$  is satisfiable.

The set  $S$  of clauses obtained by the Tseitin transformation is *equisatisfiable* with the original formula  $F$ .

- Every model of  $S$  is a model of  $F$ .
- Every model of  $F$  can be extended to a model of  $S$  by assigning the values of fresh variables according to their definitions.

## Avoiding the Exponential Blowup

**Example:**  $F = p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))$

Applying the Tseitin transformation gives us:

$$\begin{aligned} S = & p_1 \leftrightarrow (p_2 \leftrightarrow n_3) \wedge \\ & n_3 \leftrightarrow (p_3 \leftrightarrow n_4) \wedge \\ & n_4 \leftrightarrow (p_4 \leftrightarrow n_5) \wedge \\ & n_5 \leftrightarrow (p_5 \leftrightarrow p_6) \end{aligned}$$

The equivalent CNF of  $F$  consists of  $2^5 = 32$  clauses, and grows exponentially with number of variables.

The equisatisfiable CNF of  $F$  consists of 16 clauses, yet introduces 3 fresh variables, and grows linearly with the number of variables.



## Clausal Form

**Definition 10** (Clausal form): A *clausal form* of a formula  $F$  is a set  $S_F$  of clauses which is satisfiable iff  $F$  is satisfiable.

A clausal form of a *set* of formulas  $S$  is a set  $S'$  of clauses which is satisfiable iff  $S$  is satisfiable.

Even stronger requirement:

- $F$  and  $S_F$  have the same models in the language of  $F$ .
- $S$  and  $S'$  have the same models in the language of  $S$ .

The main advantage of the clausal form over the equivalent CNF is that we can convert any formula into a set of clauses in *almost linear time*.

1. If  $F$  is a formula which has the form  $C_1 \wedge \dots \wedge C_n$ , where  $n > 0$  and each  $C_i$  is a clause, then its clausal form is  $S \stackrel{\text{def}}{=} \{C_1, \dots, C_n\}$ .
2. Otherwise, apply Tseitin transformation: introduce a name for each subformula  $A$  of  $F$  such that  $A$  is not a literal and use this name instead of a subformula  $A$ .

# TODO

---

- ☐ Exercises
- ☐ Example: convert formula to clausal form
- ☐ DNF vs CNF satisfiability