1.5 Complexity Theory Prerequisites

A *decision problem* is a subset $L \subseteq \Sigma^*$ for some fixed finite alphabet Σ.

The function $\mathrm{chr}(L, x)$ denotes the *characteristic function* for some decision problem L and is defined by $\mathrm{chr}(L, u) = 1$ if $u \in L$ and $\mathrm{chr}(L, u) = 0$ otherwise.

P and NP

A decision problem is called *solvable in polynomial time* if its characteristic function can be computed in polynomial time. The class P denotes all polynomial-time decision problems.

We say that a decision problem L is in *NP* if there is a predicate $Q(x, y)$ and a polynomial $p(n)$ such that for all $u \in \Sigma^*$ we have

(i) $u \in L$ if and only if there is a $v \in \Sigma^*$ with $|v| \leq p(|u|)$ and $Q(u, v)$ holds, and

(ii) the predicate Q is in P.

Reducibility, NP-Hardness, NP-Completeness

A decision problem L is *polynomial-time reducible* to a decision problem L' if there is a function g computable in polynomial time such that for all $u \in \Sigma^*$ we have $u \in L$ iff $g(u) \in L'$.

For example, if L is polynomial-time reducible to L' and $L' \in \mathrm{P}$ then $L \in \mathrm{P}$.

A decision problem is *NP-hard* if every problem in NP is polynomial-time reducible to it.

A decision problem is *NP-complete* if it is NP-hard and in NP.

2 Propositional Logic

Propositional logic

- logic of truth values
- decidable (but NP-complete)
- can be used to describe functions over a finite domain
- industry standard for many analysis/verification tasks (e. g., model checking),
- growing importance for discrete optimization problems

2.1 Syntax

- propositional variables
- logical connectives
  $\Rightarrow$ Boolean combinations

Propositional Variables

Let Π be a set of *propositional variables*.

We use letters P, Q, R, S, to denote propositional variables.

Propositional Formulas

F_Π is the set of propositional formulas over Π defined inductively as follows:

$$
\begin{array}{lcll}
F, G & ::= & \bot & \text{(falsum)} \\
 & \mid & \top & \text{(verum)} \\
 & \mid & P, \quad P \in \Pi & \text{(atomic formula)} \\
 & \mid & (\neg F) & \text{(negation)} \\
 & \mid & (F \wedge G) & \text{(conjunction)} \\
 & \mid & (F \vee G) & \text{(disjunction)} \\
 & \mid & (F \rightarrow G) & \text{(implication)} \\
 & \mid & (F \leftrightarrow G) & \text{(equivalence)}
\end{array}
$$

Notational Conventions

As a notational convention we assume that $\neg$ binds strongest, and we remove outermost parentheses, so $\neg P \vee Q$ is actually a shorthand for $((\neg P) \vee Q)$.

Instead of $((P \wedge Q) \wedge R)$ we simply write $P \wedge Q \wedge R$ (and analogously for $\vee$).

For all other logical connectives we will use parentheses when needed.

Formula Manipulation

Automated reasoning is very much formula manipulation. In order to precisely represent the manipulation of a formula, we introduce positions.

A *position* is a word over $\mathbb{N}$. The set of positions of a formula F is inductively defined by

$$\begin{aligned}
\operatorname{pos}(F) &:= \{\varepsilon\} \text{ if } F \in \{\top, \bot\} \text{ or } F \in \Pi \\
\operatorname{pos}(\neg F) &:= \{\varepsilon\} \cup \{\, 1p \mid p \in \operatorname{pos}(F)\,\} \\
\operatorname{pos}(F \circ G) &:= \{\varepsilon\} \cup \{\, 1p \mid p \in \operatorname{pos}(F)\,\} \cup \{\, 2p \mid p \in \operatorname{pos}(G)\,\} \\
&\quad\ \text{where } \circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}.
\end{aligned}$$

The prefix order $\leq$ on positions is defined by $p \leq q$ if there is some p' such that $pp' = q$.

Note that the prefix order is partial, e.g., the positions 12 and 21 are not comparable, they are "parallel", see below.

By $<$ we denote the strict part of $\leq$, that is, $p < q$ if $p \leq q$ but not $q \leq p$.

By $\parallel$ we denote incomparable positions, that is, $p \parallel q$ if neither $p \leq q$ nor $q \leq p$.

We say that p is *above* q if $p \leq q$, p is *strictly above* q if $p < q$, and p and q are *parallel* if $p \parallel q$.

The *size* of a formula F is given by the cardinality of $\operatorname{pos}(F)$: $|F| := |\operatorname{pos}(F)|$.

The *subformula* of F at position $p \in \operatorname{pos}(F)$ is recursively defined by

$$\begin{aligned}
F|_\varepsilon &:= F \\
(\neg F)|_{1p} &:= F|_p \\
(F_1 \circ F_2)|_{ip} &:= F_i|_p \quad \text{where } i \in \{1, 2\} \\
&\qquad\qquad \text{and } \circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}.
\end{aligned}$$

Finally, the *replacement* of a subformula at position $p \in \text{pos}(F)$ by a formula G is recursively defined by

$$
\begin{aligned}
F[G]_\varepsilon &:= G \\
(\neg F)[G]_{1p} &:= \neg(F[G]_p) \\
(F_1 \circ F_2)[G]_{1p} &:= (F_1[G]_p \circ F_2) \\
(F_1 \circ F_2)[G]_{2p} &:= (F_1 \circ F_2[G]_p) \\
&\qquad \text{where } \circ \in \{\land, \lor, \rightarrow, \leftrightarrow\}.
\end{aligned}
$$

Example 2.1 *The set of positions for the formula* $F = (P \rightarrow Q) \rightarrow (P \land \neg Q)$ *is* $\text{pos}(F) = \{\varepsilon, 1, 11, 12, 2, 21, 22, 221\}$.

The subformula at position 22 *is* $F|_{22} = \neg Q$ *and replacing this formula by* $P \leftrightarrow Q$ *results in* $F[P \leftrightarrow Q]_{22} = (P \rightarrow Q) \rightarrow (P \land (P \leftrightarrow Q))$.

Polarities

A further prerequisite for efficient formula manipulation is the polarity of a subformula G of F. The polarity determines the number of "negations" starting from F down to G. It is 1 for an even number, -1 for an odd number and 0 if there is at least one equivalence connective along the path.

The *polarity* of a subformula $G = F|_p$ at position p is $\text{pol}(F, p)$, where pol is recursively defined by

$$
\begin{aligned}
\text{pol}(F, \varepsilon) &:= 1 \\
\text{pol}(\neg F, 1p) &:= -\text{pol}(F, p) \\
\text{pol}(F_1 \circ F_2, ip) &:= \text{pol}(F_i, p) \text{ if } \circ \in \{\land, \lor\} \\
\text{pol}(F_1 \rightarrow F_2, 1p) &:= -\text{pol}(F_1, p) \\
\text{pol}(F_1 \rightarrow F_2, 2p) &:= \text{pol}(F_2, p) \\
\text{pol}(F_1 \leftrightarrow F_2, ip) &:= 0
\end{aligned}
$$

Example 2.2 *Let* $F = (P \rightarrow Q) \rightarrow (P \land \neg Q)$. *Then* $\text{pol}(F, 1) = \text{pol}(F, 12) = \text{pol}(F, 221) = -1$ *and* $\text{pol}(F, \varepsilon) = \text{pol}(F, 11) = \text{pol}(F, 2) = \text{pol}(F, 21) = \text{pol}(F, 22) = 1$.

For the formula $F' = (P \land Q) \leftrightarrow (P \lor Q)$ *we get* $\text{pol}(F', \varepsilon) = 1$ *and* $\text{pol}(F', p) = 0$ *for all* $p \in \text{pos}(F')$ *different from* ε.

2.2 Semantics

In *classical logic* (dating back to Aristotle) there are "only" two truth values "true" and "false" which we shall denote, respectively, by 1 and 0.

There are *multi-valued logics* having more than two truth values.

Valuations

A propositional variable has no intrinsic meaning. The meaning of a propositional variable has to be defined by a valuation.

A Π-*valuation* is a map

$$\mathcal{A} : \Pi \to \{0, 1\}.$$

where $\{0, 1\}$ is the set of *truth values*.

Truth Value of a Formula in $\mathcal{A}$

Given a Π-valuation $\mathcal{A}$, its extension to formulas $\mathcal{A}^* : \mathrm{F}_\Pi \to \{0, 1\}$ is defined inductively as follows:

$$\begin{aligned}
\mathcal{A}^*(\bot) &= 0 \\
\mathcal{A}^*(\top) &= 1 \\
\mathcal{A}^*(P) &= \mathcal{A}(P) \\
\mathcal{A}^*(\neg F) &= 1 - \mathcal{A}^*(F) \\
\mathcal{A}^*(F \land G) &= \min(\mathcal{A}^*(F), \mathcal{A}^*(G)) \\
\mathcal{A}^*(F \lor G) &= \max(\mathcal{A}^*(F), \mathcal{A}^*(G)) \\
\mathcal{A}^*(F \to G) &= \max(1 - \mathcal{A}^*(F), \mathcal{A}^*(G)) \\
\mathcal{A}^*(F \leftrightarrow G) &= \text{if } \mathcal{A}^*(F) = \mathcal{A}^*(G) \text{ then 1 else 0}
\end{aligned}$$

For simplicity, the extension $\mathcal{A}^*$ of $\mathcal{A}$ is usually also denoted by $\mathcal{A}$.

2.3 Models, Validity, and Satisfiability

Let F be a Π-formula.

We say that F is *true* under $\mathcal{A}$ ($\mathcal{A}$ is a *model* of F; F *is valid* in $\mathcal{A}$; F *holds* under $\mathcal{A}$), written $\mathcal{A} \models F$, if $\mathcal{A}(F) = 1$.

We say that F is *valid* or that F is a *tautology*, written $\models F$, if $\mathcal{A} \models F$ for all Π-valuations $\mathcal{A}$.

F is called *satisfiable* if there exists an $\mathcal{A}$ such that $\mathcal{A} \models F$. Otherwise F is called *unsatisfiable* (or *contradictory*).

Entailment and Equivalence

F *entails* (*implies*) G (or G *is a consequence of* F), written $F \models G$, if for all Π-valuations $\mathcal{A}$ we have

$$\text{if } \mathcal{A} \models F \text{ then } \mathcal{A} \models G,$$

or equivalently

$$\mathcal{A}(F) \leq \mathcal{A}(G).$$

F and G are called *equivalent*, written $F \models\!\mid G$, if for all Π-valuations $\mathcal{A}$ we have

$$\mathcal{A} \models F \text{ if and only if } \mathcal{A} \models G,$$

or equivalently

$$\mathcal{A}(F) = \mathcal{A}(G).$$

Proposition 2.3 *$F \models G$ if and only if $\models (F \rightarrow G)$.*

Proof. ($\Rightarrow$) Suppose that F entails G. Let $\mathcal{A}$ be an arbitrary Π-valuation. We have to show that $\mathcal{A} \models F \rightarrow G$. If $\mathcal{A}(F) = 1$, then $\mathcal{A}(G) = 1$ (since $F \models G$), and hence $\mathcal{A}(F \rightarrow G) = \max(1-1, 1) = 1$. Otherwise $\mathcal{A}(F) = 0$, then $\mathcal{A}(F \rightarrow G) = \max(1-0, \mathcal{A}(G)) = 1$ independently of $\mathcal{A}(G)$. In both cases, $\mathcal{A} \models F \rightarrow G$.

($\Leftarrow$) Suppose that F does not entail G. Then there exists a Π-valuation $\mathcal{A}$ such that $\mathcal{A} \models F$, but not $\mathcal{A} \models G$. Consequently, $\mathcal{A}(F \rightarrow G) = \max(1 - \mathcal{A}(F), \mathcal{A}(G)) = \max(1-1, 0) = 0$, so $(F \rightarrow G)$ does not hold under $\mathcal{A}$. $\square$

Proposition 2.4 *$F \models\mid G$ if and only if $\models (F \leftrightarrow G)$.*

Proof. Analogously to Prop. 2.3. $\square$

Entailment is extended to sets of formulas N in the "natural way":

$N \models F$ if for all Π-valuations $\mathcal{A}$:
if $\mathcal{A} \models G$ for all $G \in N$, then $\mathcal{A} \models F$.

Note: Formulas are always finite objects; but sets of formulas may be infinite. Therefore, it is in general not possible to replace a set of formulas by the conjunction of its elements.

Validity vs. Unsatisfiability

Validity and unsatisfiability are just two sides of the same medal as explained by the following proposition.

Proposition 2.5 *F is valid if and only if $\neg F$ is unsatisfiable.*

Proof. ($\Rightarrow$) If F is valid, then $\mathcal{A}(F) = 1$ for every valuation $\mathcal{A}$. Hence $\mathcal{A}(\neg F) = 1 - \mathcal{A}(F) = 0$ for every valuation $\mathcal{A}$, so $\neg F$ is unsatisfiable.

($\Leftarrow$) Analogously. $\square$

Hence in order to design a theorem prover (validity checker) it is sufficient to design a checker for unsatisfiability.

In a similar way, entailment can be reduced to unsatisfiability and vice versa:

Proposition 2.6 *$N \models F$ if and only if $N \cup \{\neg F\}$ is unsatisfiable.*

Proposition 2.7 *$N \models \bot$ if and only if N is unsatisfiable.*

Checking Unsatisfiability

Every formula F contains only finitely many propositional variables. Obviously, $\mathcal{A}(F)$ depends only on the values of those finitely many variables in F under $\mathcal{A}$.

If F contains n distinct propositional variables, then it is sufficient to check 2^n valuations to see whether F is satisfiable or not $\Rightarrow$ truth table.

So the satisfiability problem is clearly decidable (but, by Cook's Theorem, NP-complete).

Nevertheless, in practice, there are (much) better methods than truth tables to check the satisfiability of a formula. (later more)

Substitution Theorem

Proposition 2.8 *Let $\mathcal{A}$ be a valuation, let F and G be formulas, and let $H = H[F]_p$ be a formula in which F occurs as a subformula at position p.*

If $\mathcal{A}(F) = \mathcal{A}(G)$, then $\mathcal{A}(H[F]_p) = \mathcal{A}(H[G]_p)$.

Proof. The proof proceeds by induction over the length of p.

If $p = \varepsilon$, then $H[F]_\varepsilon = F$ and $H[G]_\varepsilon = G$, so $\mathcal{A}(H[F]_p) = \mathcal{A}(F) = \mathcal{A}(G) = H[G]_p$ by assumption.

If $p = 1q$ or $p = 2q$, then $H = \neg H_1$ or $H = H_1 \circ H_2$ for $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$. Assume that $p = 1q$ and that $H = H_1 \wedge H_2$, hence $H[F]_p = H[F]_{1q} = H_1[F]_q \wedge H_2$. By the induction hypothesis, $\mathcal{A}(H_1[F]_q) = \mathcal{A}(H_1[G]_q)$. Hence $\mathcal{A}(H[F]_{1q}) = \mathcal{A}(H_1[F]_q \wedge H_2) = \min(\mathcal{A}(H_1[F]_q), \mathcal{A}(H_2)) = \min(\mathcal{A}(H_1[G]_q), \mathcal{A}(H_2)) = \mathcal{A}(H_1[G]_q \wedge H_2) = \mathcal{A}(H[G]_{1q})$.

The case $p = 2q$ and the other boolean connectives are handled analogously. $\square$

Theorem 2.9 *Let F and G be equivalent formulas, let $H = H[F]_p$ be a formula in which F occurs as a subformula at position p.*

Then $H[F]_p$ is equivalent to $H[G]_p$.

Proof. We have to show that $\mathcal{A}(H[F]_p) = \mathcal{A}(H[G]_p)$ for every Π-valuation $\mathcal{A}$.

Choose $\mathcal{A}$ arbitrarily. Since F and G are equivalent, we know that $\mathcal{A}(F) = \mathcal{A}(G)$. Hence, by the previous proposition, $\mathcal{A}(H[F]_p) = \mathcal{A}(H[G]_p)$. $\square$

Some Important Equivalences

Proposition 2.10 *The following equivalences hold for all formulas F, G, H:*

$$
\begin{array}{rcll}
(F \wedge F) & \equiv & F & \\
(F \vee F) & \equiv & F & \textit{(Idempotency)} \\[1ex]
(F \wedge G) & \equiv & (G \wedge F) & \\
(F \vee G) & \equiv & (G \vee F) & \textit{(Commutativity)} \\[1ex]
(F \wedge (G \wedge H)) & \equiv & ((F \wedge G) \wedge H) & \\
(F \vee (G \vee H)) & \equiv & ((F \vee G) \vee H) & \textit{(Associativity)} \\[1ex]
(F \wedge (G \vee H)) & \equiv & ((F \wedge G) \vee (F \wedge H)) & \\
(F \vee (G \wedge H)) & \equiv & ((F \vee G) \wedge (F \vee H)) & \textit{(Distributivity)} \\[1ex]
(F \wedge (F \vee G)) & \equiv & F & \\
(F \vee (F \wedge G)) & \equiv & F & \textit{(Absorption)} \\[1ex]
(\neg\neg F) & \equiv & F & \textit{(Double Negation)} \\[1ex]
\neg(F \wedge G) & \equiv & (\neg F \vee \neg G) & \\
\neg(F \vee G) & \equiv & (\neg F \wedge \neg G) & \textit{(De Morgan's Laws)} \\[1ex]
(F \wedge G) & \equiv & F, \textit{ if } G \textit{ is a tautology} & \\
(F \vee G) & \equiv & \top, \textit{ if } G \textit{ is a tautology} & \\
(F \wedge G) & \equiv & \bot, \textit{ if } G \textit{ is unsatisfiable} & \\
(F \vee G) & \equiv & F, \textit{ if } G \textit{ is unsatisfiable} & \textit{(Tautology Laws)} \\[1ex]
(F \leftrightarrow G) & \equiv & ((F \rightarrow G) \wedge (G \rightarrow F)) & \\
(F \leftrightarrow G) & \equiv & ((F \wedge G) \vee (\neg F \wedge \neg G)) & \textit{(Equivalence)} \\[1ex]
(F \rightarrow G) & \equiv & (\neg F \vee G) & \textit{(Implication)}
\end{array}
$$

2.4 Normal Forms

We define *conjunctions* of formulas as follows:

$\bigwedge_{i=1}^{0} F_i = \top$.

$\bigwedge_{i=1}^{1} F_i = F_1$.

$\bigwedge_{i=1}^{n+1} F_i = \bigwedge_{i=1}^{n} F_i \wedge F_{n+1}$.

and analogously *disjunctions:*

$\bigvee_{i=1}^{0} F_i = \bot$.

$\bigvee_{i=1}^{1} F_i = F_1$.

$\bigvee_{i=1}^{n+1} F_i = \bigvee_{i=1}^{n} F_i \vee F_{n+1}$.

Literals and Clauses

A *literal* is either a propositional variable P or a negated propositional variable $\neg P$.

A *clause* is a (possibly empty) disjunction of literals.

CNF and DNF

A formula is in *conjunctive normal form (CNF, clause normal form),* if it is a conjunction of disjunctions of literals (or in other words, a conjunction of clauses).

A formula is in *disjunctive normal form (DNF),* if it is a disjunction of conjunctions of literals.

Warning: definitions in the literature differ:

are complementary literals permitted?
are duplicated literals permitted?
are empty disjunctions/conjunctions permitted?

Checking the validity of CNF formulas or the unsatisfiability of DNF formulas is easy:

A formula in CNF is valid, if and only if each of its disjunctions contains a pair of complementary literals P and $\neg P$.

Conversely, a formula in DNF is unsatisfiable, if and only if each of its conjunctions contains a pair of complementary literals P and $\neg P$.

On the other hand, checking the unsatisfiability of CNF formulas or the validity of DNF formulas is known to be coNP-complete.

Conversion to CNF/DNF

Proposition 2.11 *For every formula there is an equivalent formula in CNF (and also an equivalent formula in DNF).*

Proof. We describe a (naive) algorithm to convert a formula to CNF.

Apply the following rules as long as possible (modulo commutativity of $\wedge$ and $\vee$):

Step 1: Eliminate equivalences:

$$H[F \leftrightarrow G]_p \;\Rightarrow_{\mathrm{CNF}}\; H[(F \rightarrow G) \wedge (G \rightarrow F)]_p$$

Step 2: Eliminate implications:

$$H[F \rightarrow G]_p \;\Rightarrow_{\mathrm{CNF}}\; H[\neg F \vee G]_p$$

Step 3: Push negations downward:

$$\begin{aligned} H[\neg(F \vee G)]_p &\;\Rightarrow_{\mathrm{CNF}}\; H[\neg F \wedge \neg G]_p \\ H[\neg(F \wedge G)]_p &\;\Rightarrow_{\mathrm{CNF}}\; H[\neg F \vee \neg G]_p \end{aligned}$$

Step 4: Eliminate multiple negations:

$$H[\neg\neg F]_p \;\Rightarrow_{\mathrm{CNF}}\; H[F]_p$$

Step 5: Push disjunctions downward:

$$H[(F \wedge F') \vee G]_p \;\Rightarrow_{\mathrm{CNF}}\; H[(F \vee G) \wedge (F' \vee G)]_p$$

Step 6: Eliminate $\top$ and $\bot$:

$$\begin{aligned} H[F \wedge \top]_p &\;\Rightarrow_{\mathrm{CNF}}\; H[F]_p \\ H[F \wedge \bot]_p &\;\Rightarrow_{\mathrm{CNF}}\; H[\bot]_p \\ H[F \vee \top]_p &\;\Rightarrow_{\mathrm{CNF}}\; H[\top]_p \\ H[F \vee \bot]_p &\;\Rightarrow_{\mathrm{CNF}}\; H[F]_p \\ H[\neg\bot]_p &\;\Rightarrow_{\mathrm{CNF}}\; H[\top]_p \\ H[\neg\top]_p &\;\Rightarrow_{\mathrm{CNF}}\; H[\bot]_p \end{aligned}$$

Proving termination is easy for steps 2, 4, and 6; steps 1, 3, and 5 are a bit more complicated.

For step 1, we can prove termination in the following way: We define a function μ_1 from formulas to positive integers such that $\mu_1(\bot) = \mu_1(\top) = \mu_1(P) = 1$, $\mu_1(\neg F) = \mu_1(F)$, $\mu_1(F \wedge G) = \mu_1(F \vee G) = \mu_1(F \rightarrow G) = \mu_1(F) + \mu_1(G)$, and $\mu_1(F \leftrightarrow G) = 2\mu_1(F) + 2\mu_1(G) + 1$. Observe that μ_1 is constructed in such a way that $\mu_1(F) > \mu_1(G)$ implies $\mu_1(H[F]) > \mu_1(H[G])$ for all formulas F, G, and H. Furthermore, μ_1 has the property that swapping the arguments of some $\wedge$ or $\vee$ in a formula F does not change the value of $\mu_1(F)$. (This is important since the transformation rules can be applied modulo commutativity of $\wedge$ and $\vee$.). Using these properties, we can show that whenever a formula H' is the result of applying the rule of step 1 to a formula H, then $\mu_1(H) > \mu_1(H')$. Since μ_1 takes only positive integer values, step 1 must terminate.

Termination of steps 3 and 5 is proved similarly. For step 3, we use function μ_2 from formulas to positive integers such that $\mu_2(\bot) = \mu_2(\top) = \mu_2(P) = 1$, $\mu_2(\neg F) = 2\mu_2(F)$, $\mu_2(F \wedge G) = \mu_2(F \vee G) = \mu_2(F \rightarrow G) = \mu_2(F \leftrightarrow G) = \mu_2(F) + \mu_2(G) + 1$. Whenever a formula H' is the result of applying a rule of step 3 to a formula H, then $\mu_2(H) > \mu_2(H')$. Since μ_2 takes only positive integer values, step 3 must terminate.

For step 5, we use a function μ_3 from formulas to positive integers such that $\mu_3(\bot) = \mu_3(\top) = \mu_3(P) = 1$, $\mu_3(\neg F) = \mu_3(F) + 1$, $\mu_3(F \wedge G) = \mu_3(F \rightarrow G) = \mu_3(F \leftrightarrow G) = \mu_3(F) + \mu_3(G) + 1$, and $\mu_3(F \vee G) = 2\mu_3(F)\mu_3(G)$. Again, if a formula H' is the result of applying a rule of step 5 to a formula H, then $\mu_3(H) > \mu_3(H')$. Since μ_3 takes only positive integer values, step 5 terminates, too.

The resulting formula is equivalent to the original one and in CNF.

The conversion of a formula to DNF works in the same way, except that conjunctions have to be pushed downward in step 5. □

Negation Normal Form (NNF)

The formula after application of Step 4 is said to be in *Negation Normal Form*, i.e., it contains neither $\rightarrow$ nor $\leftrightarrow$ and negation symbols only occur in front of propositional variables (atoms).

Complexity

Conversion to CNF (or DNF) may produce a formula whose size is *exponential* in the size of the original one.