SAPIENZA
UNIVERSITÀ DI ROMA

"Sapienza" University of Rome
Faculty of Information Engineering, Informatics and Statistics
Department of Computer Science

Mathematical Logic for Computer Science

Lecture notes integrated with the book "Introduction to Mathematical Logic", E. Mendelson

Author
Simone Bianco

February 27, 2025

Contents

Information and Contacts

Personal notes and summaries collected as part of the *Mathematical Logic for Computer Science* course offered by the degree in Computer Science of the University of Rome "La Sapienza".

Further information and notes can be found at the following link:
https://github.com/Exyss/university-notes. Anyone can feel free to report inaccuracies, improvements or requests through the Issue system provided by GitHub itself or by contacting the author privately:

- Email: **bianco.simone@outlook.it**
- LinkedIn: **Simone Bianco**

The notes are constantly being updated, so please check if the changes have already been made in the most recent version.

Suggested prerequisites:

Sufficient knowledge of logic and theoretical computer science

Licence:

1 Propositional logic

1.1 Introduction to logical constructs

Logic is concerned with the validity of reasoning independently of the meaning of its components. In particular,**propositional logic** deals with studying the properties of certain logical constructs used in natural language as well as in scientific and mathematical practice. In particular, we're interested in the following constructs:

- *Negation* ($\neg$): This operation inverts the truth value of a statement. If a statement is true, its negation is false, and vice versa.
- *Conjunction* ($\wedge$): This operation represents the logical "and". The result is true if and only if both components are true.
- *Disjunction* ($\vee$): This operation represents the logical "or". The result is true if at least one of the components is true.
- *Implication* ($\rightarrow$): This operation represents the logical "if ... then" statement. The result is false only if the first component is true and the second component is false. In other words, this operator forces that when the premise holds, the conclusion must always follow (if the premise is false, the conclusion can be either true or false).
- *Equivalence* ($\leftrightarrow$): This operation represents the logical "if and only if" statement. The result is true when both components have the same truth value, i.e. they're either both true or both false.

For instance, consider the following simple arithmetic argument:

1. If $a = 0$ or $b = 0$, then $a \cdot b = 0$
2. $a \cdot b \neq 0$
3. $a \neq 0$ and $b \neq 0$

Intuitively, the third proposition "follows" from the other two propositions, i.e. it is the conclusion of an argument with the first two as premises. To formalize this argument, we first identify those parts that cannot be further analyzed in terms of logical constructs and that can be true or false (called **atomic parts**). In our case, these atomic parts are $a = 0$, $b = 0$, and $a \cdot b = 0$.

We assign a distinct letter to each of these atomic parts: for $a = 0$, we associate A; for $b = 0$, we associate B; and for $a \cdot b = 0$, we associate C. Then, we replace the logical constructs in natural language (if ... then, or, and, not) with formal symbols, called Boolean connectives: $\neg$ for negation, $\vee$ for disjunction, $\wedge$ for conjunction, $\rightarrow$ for implication (if ... then), and $\leftrightarrow$ for equivalence (if and only if).

We obtain the following formalization:

1. $(A \vee B) \rightarrow C$
2. $\neg C$
3. $\neg A \wedge \neg B$

where we read $A \vee B$ as "A or B", $\neg C$ as "not C" and so on. Consider now the following verbal argument:

1. If the father is tall or the mother is tall, then the child is tall
2. The child is short
3. The father is short and the mother is short

If we attempt a formalization of the argument, we quickly realize that we obtain the same formalization as the previous argument, where we write A for "the father is tall", B for "the mother is tall", and C for "the child is tall". Therefore, the two arguments are identical. Formal logic allows us to identify the common logical structure in arguments that deal with different objects and structures, as is the case in the examples above. However, in this case the first premise is obviously known to be empirically false. Nevertheless, if we assume it as premise for an argument, we intuitively recognize that the reasoning is **valid** (or correct). When we say the argument is valid (or correct), we mean that if the premises are true, then the conclusion is also true; not that the premises are true.

To evaluate the truthfulness of a propositional statement, we use **truth tables**.In logic, a truth table is a table used to determine whether a logical expression is true or false based on the **truth values** of its components. Truth tables are primarily used to evaluate the validity of logical propositions and determine the relationships between various logical statements. Each row of a truth table represents a unique combination of truth values for the variables involved in the logical expression. The columns show the intermediate truth values of the components of the expression, ultimately leading to the evaluation of the entire logical formula.

A	$\neg A$
0	1
1	0

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

A	B	$A \to B$
0	0	1
0	1	1
1	0	0
1	1	1

A	B	$A \leftrightarrow B$
0	0	1
0	1	0
1	0	0
1	1	1

Figure 1.1: Truth tables of the five logical operators previously introduces. The value 0 represents the falseness of the propositions, while the value 1 represents their truthfulness.

Consider now a more complex logical expression $A \to (B \wedge C)$. A truth table for this expression would involve evaluating the truth values for A, B and C, then determining the truth value of the conjunction $B \wedge C$, and finally computing the implication $A \to (B \wedge C)$.

A	B	C	$B \wedge C$	$A \to (B \wedge C)$
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Figure 1.2: Truth table for the logical expression $A \to (B \wedge C)$

1.2 Mathematical formalization

The first step is to define a completely rigorous formal language that can be subjected to mathematical study. For propositional logic, the procedure is simple: propositions are formalized by formulas obtained by applying Boolean connectives to some atomic building blocks, called *propositional variables*. These variables intuitively represent propositions that cannot be further analyzed in terms of logical operators such as negation, conjunction, disjunction, implication, or equivalence.

Definition 1: Language and propositions

A propositional language is a set $\mathcal{L} = \{p_1, \ldots, p_n\}$, where each p_i is a symbol called propositional variable. Given a propositional language $\mathcal{L}$, the set of propositions (or valid formulas) over $\mathcal{L}$, denoted with $\mathsf{PROP}_\mathcal{L}$ (or $\mathsf{FML}_\mathcal{L}$) is inductively defined as:

- Each propositional variable in $\mathcal{L}$ is a proposition, i.e. $\mathcal{L} \subseteq \mathsf{PROP}_\mathcal{L}$
- If $A \in \mathsf{PROP}_\mathcal{L}$ then $\neg A \in \mathsf{PROP}_\mathcal{L}$
- If $A, B \in \mathsf{PROP}_\mathcal{L}$ then $(A \wedge B), (A \vee B), (A \to B), (A \leftrightarrow B) \in \mathsf{PROP}_\mathcal{L}$

When $\mathcal{L}$ is non-ambiguous in the context, we denote $\mathsf{PROP}_\mathcal{L}$ directly as PROP. Moreover, to make expressions easier to read, we use the following precedence rules: $\neg$ has precedence over $\wedge, \vee$, while $\wedge, \vee$ have precedence over $\to, \leftrightarrow$. In other words, we have that $(\neg((\neg A) \vee B)) \to C$ can be written as $\neg(\neg A \vee B) \to C$.

After formalizing the concept of propositional language and propositions, we're ready to formalize the concept of truth tables. Each row of a truth table can be described through **assignments**. Here, an assignment is any function $\alpha : \mathcal{L} \to \{0, 1\}$, i.e. a map that assigns a truth value to each propositional variable of the language. By extension, any assignment over $\mathcal{L}$ induces an assignment over PROP. These assignments are functions $\widehat{\alpha} : \mathsf{PROP} \to \{0, 1\}$ inductively defined as:

$$\widehat{\alpha}(A) = \alpha(A) \text{ when } A \in \mathcal{L}$$

$$\widehat{\alpha}(\neg A) = \begin{cases} 1 & \text{if } \widehat{\alpha}(A) = 0 \\ 0 & \text{if } \widehat{\alpha}(A) = 1 \end{cases}$$

$$\widehat{\alpha}(A \wedge B) = \begin{cases} 1 & \text{if } \widehat{\alpha}(A) = \widehat{\alpha}(B) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\widehat{\alpha}(A \vee B) = \begin{cases} 0 & \text{if } \widehat{\alpha}(A) = \widehat{\alpha}(B) = 0 \\ 1 & \text{otherwise} \end{cases}$$

$$\widehat{\alpha}(A \to B) = \begin{cases} 0 & \text{if } \widehat{\alpha}(A) = 1 \text{ and } \widehat{\alpha}(B) = 0 \\ 1 & \text{otherwise} \end{cases}$$

$$\widehat{\alpha}(A \leftrightarrow B) = \begin{cases} 1 & \text{if } \widehat{\alpha}(A) = \widehat{\alpha}(B) \\ 0 & \text{otherwise} \end{cases}$$

To make notation easier, we'll directly refer to $\widehat{\alpha}$ as α. Using the concept of assignments, we can also define the concept of **logical equivalence**. Informally, two formulas are said to be logically equivalent when they share the same exact truth table.

Definition 2: Logical equivalence

We say that two formulas $A, B \in \mathsf{PROP}$ are logically equivalent, denoted with $A \equiv B$, when for every assignment α it holds that $\alpha(A) = \alpha B$.

By definition, it clearly holds that $\equiv$ is an equivalence relation since it is reflexive, symmetric and transitive. Logical equivalences allow us to treat logical components as if they were algebraic operators. For instance, common algebraic properties include:

- *Involution*: $\neg\neg A \equiv A$
- *Idempotency*: $A \vee A \equiv A$ and $A \wedge A \equiv A$
- *Associativity*: $(A \vee B) \vee C \equiv A \vee (B \vee C)$ and $(A \wedge B) \wedge C \equiv A \wedge (B \wedge C)$
- *Commutativity*: $A \vee B \equiv B \vee A$ and $A \wedge B \equiv B \wedge A$
- *Distributivity*: $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$ and $A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C)$
- *De Morgan's law*: $\neg(A \vee B) \equiv \neg A \wedge \neg B$ and $\neg(A \wedge B) \equiv \neg A \vee \neg B$

Through these basic properties, we notice that each connective can be rewritten in terms of the other connectives. In fact, we could define propositional logic using only $\neg$ and $\wedge$ (or $\neg$ and $\vee$). This property is very useful in circuit logic.

- $A \wedge B \equiv \neg(\neg A \vee \neg B)$
- $A \vee B \equiv \neg(\neg A \wedge \neg B)$
- $A \to B \equiv \neg A \vee B$
- $A \leftrightarrow B \equiv (A \to B) \wedge (B \to A)$

If for an assignment α and a formula A it holds that $\alpha(A) = 1$, we say that α *satisfies* A. If A has at least one satisfying assignment, we say that A is **satisfiable**. When no satisfying assignment exists, we say that the formula A is **unsatisfiable**. On the contrary, when every possible assignment satisfies A we say that A is a **tautology** (or *logical truth*, *logically valid*). By definition, tautologies represent statements that are always true, no matter the situation. We denote with $\mathsf{SAT}, \mathsf{UNSAT}$ and TAUT the sets of satisfiable, unsatisfiable and tautological formulas. By definition, we have that $F \in \mathsf{TAUT}$ if and only if $\neg F \in \mathsf{UNSAT}$. If a formula is not a tautology, we cannot ensure the truthfulness of the statement. Hence, we consider *true statements* only as formulas that are *always true* (the set TAUT), while we consider *false statement* as formulas that are *not always true* (the set $\mathsf{SAT} - \mathsf{TAUT}$).

With a slight abuse of notation, we define 1 and 0 as the formulas such that for every assignment it holds that $\alpha(1) = 1$ and $\alpha(0) = 0$. This allows us to further simplify logical equivalences algebraically:

- *Absorption law*: $A \vee 0 \equiv A$ and $A \wedge 1 \equiv A$
- *Cancellation law*: $A \vee 1 \equiv 1$ and $A \wedge 0 \equiv 0$
- *Tertium non datur*: $A \vee \neg A \equiv 1$ and $A \wedge \neg A \equiv 0$

Using all the algebraic rules that we have shown, we can easily evaluate propositional formulas by first reducing them to a smaller equivalent formula and then (eventually) by evaluating the truth table of the reduced form:

$$\begin{aligned} A \vee B \to B \vee \neg C &\equiv \neg(A \vee B) \vee B \vee \neg C \\ &\equiv (\neg A \wedge \neg B) \vee B \vee \neg C \\ &\equiv ((\neg A \vee B) \wedge (\neg B \vee B)) \vee \neg C \\ &\equiv ((\neg A \vee B) \wedge 1) \vee \neg C \\ &\equiv \neg A \vee B \vee \neg C \end{aligned}$$

The concepts of tautology and unsatisfiability are strictly related with the concept of **logical consequence** that we have discussed in the previous section.

Definition 3: Logical consequence

Given the formulas $A_1, \ldots, A_n, A$, we say that A is a logical consequence of $A_1, \ldots, A_n$, written as $A_1, \ldots, A_n \models A$ if whenever $A_1, \ldots, A_n$ are true A is also true.

Through truth tables, evaluating if we can conclude A through $A_1, \ldots, A_n$ is pretty easy: $A_1, \ldots, A_n \models A$ if and only if for every row where $A_1, \ldots, A_n$ are evaluated as 1 we also have that A is evaluated as 1. Since the number of rows in a truth table is exponential with respect to the number of propositional variables, enumerating the whole truth table is too much expensive. To help with this, the following theorem comes in handy.

Theorem 1

Given the formulas $A_1, \ldots, A_n, A$, the three following statements are equivalent:

1. $A_1, \ldots, A_n \models A$
2. $(A_1 \wedge \ldots \wedge A_n \to A) \in \mathsf{TAUT}$
3. $(A_1 \wedge \ldots \wedge A_n \wedge \neg A) \in \mathsf{UNSAT}$

Proof. Omitted (trivial) □

Hence, in order to evaluate if $A_1, \ldots, A_n \models A$ holds, we can show that $(A_1 \wedge \ldots \wedge A_n \to A) \equiv 1$ or that $(A_1 \wedge \ldots \wedge A_n \wedge \neg A) \equiv 0$ through the algebraic properties shown before (this may be as tedious as enumerating the whole truth table, but it is usually faster). Currently, there are no efficient algorithms to answer questions such as "is F satisfiable?", "is F unsatisfiable?" and "is F a tautology?". Finding such algorithm or proving that such algorithm cannot exist is equivalent to solving the Millennium Problem $\mathsf{P} \stackrel{?}{=} \mathsf{NP}$, which asks the question "can every efficiently verifiable problem also be efficiently solved?". For this problem, the Clay Mathematical Institute offers a prize of one million dollars. In many cases, it is possible to decide whether a certain proposition is a tautology or not, or whether a certain conclusion is a logical consequence of other propositions without constructing the truth table, but only by reasoning rigorously at a higher level.

1.3 Disjunctive and Conjunctive Normal Forms

Every propositional formula can be rewritten in a standard way. In particular, we have two standard ways to write a formula: the **Disjuntive Normal Form (DNF)** and the **Conjunctive Normal Form (CNF)**. A formula is said to be in DNF if it is a disjuction of AND clauses. An AND *clause* is a conjunction of literals. A *literal* is a propositional variable or a negation of a propositional variable. In other words, we say that F is in DNF if:

$$F = \bigvee_{j=1}^{m} D_j = \bigvee_{j=1}^{m} (\ell_{1,j} \wedge \ell_{2,j} \wedge \ldots \wedge \ell_{k_j,j})$$

where each D_j is an AND clause of the formula and each ℓ_{i_h} is a literal. Similarly, a formula is said to be in CNF if it is a conjunction of OR clauses.

$$F = \bigwedge_{j=1}^{m} C_j = \bigwedge_{j=1}^{m} (\ell_{1,j} \vee \ell_{2,j} \vee \ldots \vee \ell_{k_j,j})$$

Using the logical equivalence properties, it's clear that every formula has an equivalent DNf and CNF formula. In a more direct way, we can use truth tables (hence assignments) to yield an equivalent DNF or CNF formula.

Theorem 2: Equivalent DNF and CNF formulas

For every formula F there is a DNF formula F^{DNF} and a CNF formula F^{CNF} such that $F \equiv F^{\mathsf{DNF}} \equiv F^{\mathsf{CNF}}$.

Proof. Given $\mathcal{L} = \{p_1, \ldots, p_n\}$, consider the truth table of F.

p_1	p_2	$\ldots$	p_n	F
0	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$\vdots$	$\ddots$	$\ddots$	$\vdots$	$\vdots$
0	$\ddots$	$\ddots$	$\vdots$	$\vdots$
1	$\ddots$	$\ddots$	$\vdots$	$\vdots$
$\vdots$	$\ddots$	$\ddots$	$\vdots$	$\vdots$
1	$\cdots$	$\cdots$	$\cdots$	$\cdots$

This truth table has 2^n rows. Each j-th row defines an assignment α_j such that if p_1 assumes value $\alpha_j(p_1)$, p_2 assumes value $\alpha_j(p_2)$ and so on then A assumes value $\alpha_j(A)$. Hence, the cases where A is true are completely described by the rows i where $\alpha_i(A) = 1$.

For each $j \in [2^n]$ and each $i \in [n]$, let $\ell_{i,j}$ be defined as:

$$\ell_{i,j} = \begin{cases} p_1 & \text{if } \alpha_j(p_i) = 1 \\ \neg p_1 & \text{if } \alpha_j(p_i) = 0 \end{cases}$$

We notice that for every assignment α it holds that:

$$\alpha(\ell_{1,j} \wedge \ldots \wedge \ell_{n,j}) = 1 \implies \alpha(\ell_{1,j}) = \alpha_j(p_i), \ldots, \alpha(\ell_{n,j}) = \alpha_j(p_n)$$

This implies that for every assignment α it holds that $\alpha(A) = 1$ if and only if for some $j \in [2^n]$ it holds that α corresponds to α_j over $p_1, \ldots, p_n$. Hence, we conclude that $F \equiv F^{\mathsf{DNF}}$, where:

$$F^{\mathsf{DNF}} = \bigvee_{j=1}^{2^n} \bigwedge_{i=1}^{n} \ell_{i,j}$$

Equivalently, we can obtain F^{CNF} proceeding in the same way by defining each $\ell'_{i,j}$ as:

$$\ell'_{i,j} = \begin{cases} \neg p_1 & \text{if } \alpha_j(p_i) = 1 \\ p_1 & \text{if } \alpha_j(p_i) = 0 \end{cases}$$

In this case, for every assignment α it holds that:

$$\alpha(\ell'_{1,j} \vee \ldots \vee \ell'_{n,j}) = 0 \implies \alpha(\ell'_{1,j}) = \alpha_j(p_i), \ldots, \alpha(\ell'_{n,j}) = \alpha_j(p_n)$$

This implies that for every assignment α it holds that $\alpha(A) = 0$ if and only if for some $j \in [2^n]$ it holds that α corresponds to α_j over $p_1, \ldots, p_n$. Hence, we conclude that $F \equiv F^{\mathsf{CNF}}$, where:

$$F^{\mathsf{CNF}} = \bigwedge_{j=1}^{2^n} \bigvee_{i=1}^{n} \ell'_{i,j}$$

□

The second part of the last proof could've also been achieved through a different argument.Consider the fact that $F \equiv \neg\neg F$. Since we know that each formula has a DNF equivalent, we know that there is a formula $(\neg F)^{\mathsf{DNF}}$ such that $\neg F \equiv (\neg F)^{\mathsf{DNF}}$. By iteratively applying De Morgan's law, we know that if $(\neg F)^{\mathsf{DNF}}$ is a DNF then $\neg(\neg F^{\mathsf{DNF}})$ has an equivalent CNF formula. Hence, we conclude that $F^{\mathsf{CNF}} = \neg(\neg F)^{\mathsf{DNF}}$ is the CNF formula such that $F \equiv F^{\mathsf{CNF}}$.