Lecture Notes for

Logic and Computability

Course Number: IND04033UF

Contact

Bettina Könighofer
Institute for Applied Information Processing and Communications (IAIK)
Graz University of Technology, Austria
bettina.koenighofer@iaik.tugraz.at

1

Propositional Logic

Logic in computer science aims to develop languages to formulate specifications and model systems in such a way that we can *reason* about them *formally*. Reasoning about situations means constructing arguments that can be formally reasoned about or whose validity can be proven rigorously.

Example. Consider the following argument: "If the plane arrives late and there are no taxis at the airport, then Alice is late for her appointment." "Alice is not late for her appointment." "The plane did arrive late." "*Therefore*, there were taxis at the airport."

We can reason intuitively that the argument is valid. If there were no taxis, Alice would have been late for the appointment since the plane has been late. Since she has not been late, it must be the case that there have been available taxis at the airport. The sentence starting with 'therefore' *logically follows* from the statements before it.

Example. Consider a different argument: "If the sun is shining and John does not have his sunscreen with him, then he will get a sunburn." "John is not sunburnt." "The sun is shining." "*Therefore*, John has his sunscreen with him."

Again, the argument is valid. Notice, that both arguments actually have the same logical structure:

$$\text{If } p \text{ and not } q, \text{ then } r. \text{ Not } r. \ p. \text{ Therefore, } q.$$

To make arguments rigorous (to allow formal reasoning over arguments), we need to develop languages that enable us to express sentences in such a way that the formulation brings out the *logical structure* of the modeled sentences. In this course, we will start with the language of propositional logic.

1.1 Declarative Sentences

Propositional logic is based on sentences that make statements, so called *declarative sentences*, or *propositions*.

Definition 1.1 (Declarative Sentence) A declarative sentence is a sentence that states a fact, or describes a state of affairs. A declarative sentence can be argued to be *true* or *false*. Hence, a declarative sentence does not ask a question, or give an order.

Example. Examples for declarative sentences are:

- 10 divided by 5 is 3.
- Mozart was born in Austria.
- Santa Clause lives at the North Pole.
- Florian is back in Austria.

Example. Examples for non-declarative sentences are:

- What is your name?
- Take the dog for a walk!
- Ready, steady, go!
- Good morning.
- May the force be with you!

1.2 Syntax of Propositional Logic

In this section, we discuss the syntax of propositional logic.

Definition 1.2 (Syntax of a formal language) The *syntax* of a formal language consists of a set of symbols and rules for combining them to form formulas of the language.

The symbols of propositional logic are the *truth symbols* $\top$("true") and $\bot$ ("false"), a countably infinite set of *propositional variable symbols* (typically denoted with by $p, q, r, \dots$), the *logical connectors* also called logical operators or Boolean connectives ($\wedge, \vee, \neg, \leftrightarrow, \rightarrow, \oplus$), and *parentheses*.

Definition 1.3 (Syntax of Propositional Logic.) A *formula* in propositional logic can be obtained by using the construction rules below, and only those, finitely many times:

- A propositional variable p, or a truth symbol $\top$, and $\bot$ are formulas.
- For any formula φ, $\neg\varphi$ is also a formula.
- For any two formulas φ and ψ, the following are also formulas: $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$, and $\varphi \leftrightarrow \psi$, $\varphi \oplus \psi$.
- For any formula φ, (φ) is also a formula.

Definition 1.4 (Syntax of Propositional Logic in Backus-Naur Form) The definition of formulas in propositional logic can also be given in the Backus-Naur

form (BNF). In that notation, the above definition reads more compactly as:

$$\varphi := <\text{automic proposition}> \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \varphi \to \varphi \mid \varphi \leftrightarrow \varphi \mid (\varphi)$$

In the remainder of this course, we will use the following terminology.

Definition 1.5 (Atom (Atomic Proposition) and Literal) An *atom* (or atomic proposition) is a propositional variable $(p, q, \dots)$, or a truth symbol $\top$ or $\bot$. A *literal* is an atom α or its negation $\neg\alpha$. A formula is a literal, or the application of a logical connective to formulas (or to one formula in case of a negation).

Exercise 1.1

Model the following sentences in propositional logic,

1. "The number k is even and greater than seven"
2. "If k is even and greater than seven, then k is not a prime number"
3. "Either k is even, or k is prime, and not both"

Solution.

We define the following atomic propositions:

- a : "the number k is even"
- b : "the number k is greater than seven"
- c : "the number k is prime"

Using this atoms, we model the sentences as follows:

- Formula for sentence 1: $a \wedge b$
- Formula for sentence 2: $a \wedge b \to \neg c$
- Formula for sentence 3: $(a \vee c) \wedge \neg(a \wedge c)$

Operator Precedence and Parentheses. We define the relative operator precedence of the logical connectives from the highest to the lowest connective as follows:

$$\text{Highest} \quad \neg \quad \wedge \quad \vee \quad \to \quad \leftrightarrow \quad \text{Lowest}$$

Additionally, we define that the connectives $\to$ and $\leftrightarrow$ are *right associative*. Therefore, the formula $p \to q \to r$ is the same formula as $p \to (q \to r)$. The connectives $\vee$ and $\wedge$ are *left associative*. The relative precedence helps to reduce the number of parentheses needed.

Exercise 1.2

For the following formulas, insert parentheses to represent the order of evaluation.

- $p \wedge q \wedge r$
- $r \to p \to r$
- $p \wedge q \wedge r \to p \to r$

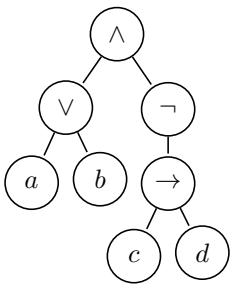
Solution.

$$\begin{aligned} p \wedge q \wedge r &\quad \text{is equivalent to} \quad (p \wedge q) \wedge r, \\ r \to p \to r &\quad \text{is equivalent to} \quad r \to (p \to r), \text{ and} \\ p \wedge q \wedge r \to p \to r &\quad \text{is equivalent to} \quad ((p \wedge q) \wedge r) \to (p \to r). \end{aligned}$$

Parse Trees

Parse trees are a simple way to check whether a string is a formula in propositional logic. A formula φ is a formula in propositional logic if all leaves are atomic propositions and all other nodes are labeled with logical operators.

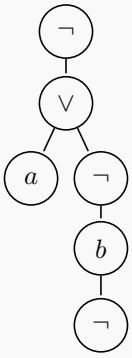
Example. The parse tree for $(a \vee b) \wedge (\neg(c \to d))$ is given below. Since all leaves are labeled with atomic propositions and all inner nodes are labeled with logical operators, the string is a formula.



Exercise 1.3

Consider the string $\neg(a \vee \neg b\neg)$. Determine whether the string forms a formula in propositional logic using its parse tree.

Solution.



The string is *not* a logical formula because of the leaf note that represents a negation and the inner node that represents the propositional variable b.

1.3 Semantics of Propositional Logic

The semantics assigns a *meaning* to a formula. In propositional logic, it assigns a truth value $\top$ or $\bot$ to formulas. We first define models, to assign truth values to propositional variables. Next, we give an inductive definition of the semantics in propositional logic.

Models

A model assigns a truth value to the propositional variables in a formula. Models are also called *valuations*, *interpretations*, or *assignments*.

Definition 1.6 (Model) A *model* $\mathcal{M}$ of a propositional formula φ is an assignment of truth values to variables in the formula. A model $\mathcal{M}$ is called a *full* (or *total*) assignment if $\mathcal{M}$ assigns a truth value to each variable in the formula. Otherwise, $\mathcal{M}$ is called a *partial* model. Thus, a partial model assigns values to some but not all variables in φ.

Let $\mathcal{M}$ be a model for a formula φ. We use the following notation:

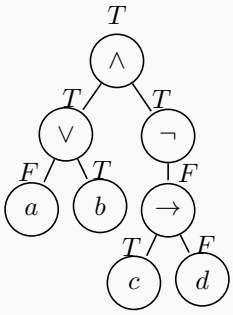
- $\varphi^{\mathcal{M}}$: The formula φ is evaluated under the model $\mathcal{M}$.
- $\mathcal{M} \vDash \varphi$: The model satisfies the formula.
- $\mathcal{M} \nvDash \varphi$: The model does not satisfy the formula.

We call a model $\mathcal{M}$ *satisfying* for a formula φ if $\varphi^{\mathcal{M}}$ evaluates to *true* or *falsifying* if $\varphi^{\mathcal{M}}$ resolves to *false*. We call $\vDash$ the *semantic entailment* relation.

Exercise 1.4

Let $\varphi = (a \vee b) \wedge (\neg(c \to d))$ and $\mathcal{M} = \{a = F,\ b = T,\ c = T,\ d = F\}$.
Use the parse tree to evaluate $\varphi^{\mathcal{M}}$.

Solution.

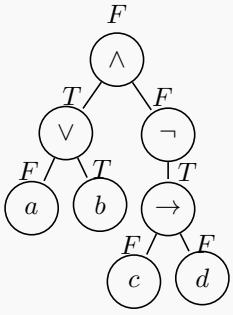


We have $\varphi^{\mathcal{M}_1} = T$, or $\mathcal{M}_1 \vDash \varphi$.

Exercise 1.5

Let $\varphi = (a \vee b) \wedge (\neg(c \to d))$ and model $\mathcal{M} = \{a = F,\ b = T,\ c = F,\ d = F\}$.
Use the parse tree to evaluate $\varphi^{\mathcal{M}}$.

Solution.



We have $\varphi^{\mathcal{M}} = F$, or $\mathcal{M} \nvDash \varphi$.

Inductive Definition of the Semantics

Base Cases. The base cases define the semantic entailment relation for the truth values (the formula $\top$ is true for any model $\mathcal{M}$, and the formula $\bot$ is false for any $\mathcal{M}$). Furthermore, it assigns the truth values to the propositional variables according to their assignment in $\mathcal{M}$.

- $\mathcal{M} \vDash \top$
- $\mathcal{M} \nvDash \bot$
- $\mathcal{M} \vDash p$ iff $\mathcal{M}[p] = \top$ ...p has the value $\top$ if $\mathcal{M}$ assigns the value $\top$ to p
- $\mathcal{M} \nvDash p$ iff $\mathcal{M}[p] = \bot$ ...p has the value $\bot$ if $\mathcal{M}$ assigns the value $\bot$ to p

Inductive Step. Assume that the formulas φ and ψ have already truth values assigned. Using φ and ψ, we can inductively assign truth values to composed formulas.

- $\mathcal{M} \vDash \neg\varphi$ iff $\mathcal{M} \nvDash \varphi$
- $\mathcal{M} \vDash \varphi \wedge \psi$ iff $\mathcal{M} \vDash \varphi$ and $\mathcal{M} \vDash \psi$
- $\mathcal{M} \vDash \varphi \vee \psi$ iff $\mathcal{M} \vDash \varphi$ or $\mathcal{M} \vDash \psi$
- $\mathcal{M} \vDash \varphi \rightarrow \psi$ iff if $\mathcal{M} \vDash \varphi$ then $\mathcal{M} \vDash \psi$
- $\mathcal{M} \vDash \varphi \leftrightarrow \psi$ iff $\mathcal{M} \vDash \varphi$ and $\mathcal{M} \vDash \psi$ or $\mathcal{M} \nvDash \varphi$ and $\mathcal{M} \nvDash \psi$.

Definition of the Semantics via Truth Tables

The semantics of the logical operators can also be defined via truth tables. The base cases are defined as before. The inductive step is given by the truth tables that define the meaning of the logical operators as given below.

φ	ψ	$\varphi \wedge \psi$	$\varphi \vee \psi$	$\varphi \rightarrow \psi$	$\varphi \leftrightarrow \psi$
F	F	F	F	T	T
F	T	F	T	T	F
T	F	F	T	F	F
T	T	T	T	T	T

Figure 1.1: Truth tables for $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$.

φ	$\neg\varphi$
F	T
T	F

Figure 1.2: Truth table for negation.

1.4 Semantic Entailment, Equivalence, Satisfiability, and Validity

Next, we discuss elementary concepts of semantics in propositional logic: the concepts of semantic entailment and semantic equivalence of two propositional formals, as well as validity and satisfiability of a propositional formula.

Definition 1.7 (Satisfiability (SAT)) A formula φ is satisfiable if and only if there exists a model $\mathcal{M}$ that satisfies φ, i.e., $\mathcal{M} \vDash \varphi$.

Definition 1.8 (Unsatisfiability (UNSAT)) A formula φ is unsatisfiable if and only if for all model $\mathcal{M}$ it holds that $\mathcal{M} \nvDash \varphi$. Therefore, no model makes the formula true. An unsatisfiable formula is also called a *contratiction*.

Definition 1.9 (Validity) A formula φ is valid if and only if for all model $\mathcal{M}$ it holds that $\mathcal{M} \vDash \varphi$. A valid formula is also called a *tautology*.

Duality Satisfiability - Validity. Satisfiability and validity are dual concepts. Therefore, it holds that

- A formula φ is valid if and only if $\neg\varphi$ is unsatisfiability.
- A formula φ is unsatisfiability if and only if $\neg\varphi$ is valid.

Definition 1.10 (Semantic Entailment) A formula φ *semantically entails* a formula ψ, if and only if every model $\mathcal{M}$ that satisfies φ ($\mathcal{M} \vDash \varphi$) also satisfies ψ ($\mathcal{M} \vDash \psi$).

Example. Consider the following examples for the semantic entailment relation between formulas.

- $a \vDash a \vee b$,
- $a \vee b \nvDash a$,
- $(p \wedge q) \vDash p$, and
- $(p \vee q) \nvDash p$.

In particular, it holds that

- $\bot \vDash \varphi$, and
- $\psi \vDash \top$.

The definition of semantic entailment requires that all models that satisfy the entailing formula also satisfy the entailed formula. $\bot \vDash \varphi$ holds since a contradiction does not have any satisfying models, a contradiction entails any possible formula. $\psi \vDash \top$ holds since all possible models satisfy a tautology. Thus, any formula entails $\top$.

Figure 1.3 gives a visual representation of the semantic entailment relation. In the illustration, we have the following area encodings.

- Big circles represent the sets of all possible models.
- Dark gray areas represent the models that satisfy the formula φ.

- Light gray areas represent the models that satisfy the formula ψ.
- Gridded areas represent the models that satisfy formulas φ and ψ.
- White areas represent the models that neither satisfy formula φ nor ψ.

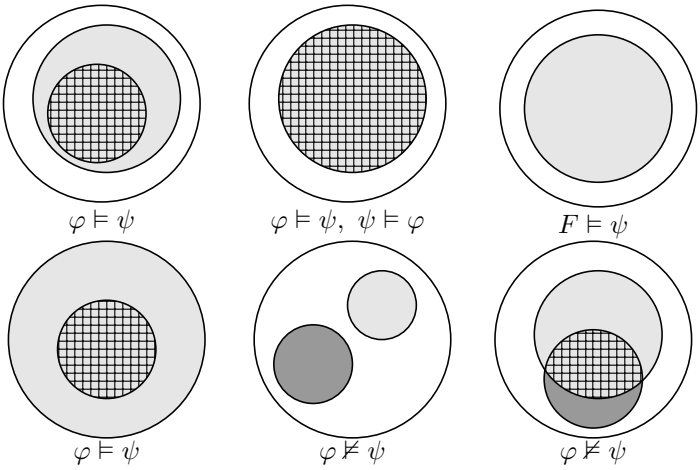


Figure 1.3: Visualisation of the semantic entailment relation.

Definition 1.11 (Semantic Equivalence) φ and ψ are semantically equivalent if and only if both evaluate to true under the same models, i.e., $\varphi \vDash \psi$ and $\psi \vDash \varphi$ holds. We denote equivalence of φ and ψ via $\varphi \equiv \psi$.

Example. The following formulas are equivalent.

- $a \to b \equiv \neg b \to \neg a$, and
- $a \to b \equiv \neg a \vee b$.

The notions of validity, satisfiability, semantic entailment, and semantic equivalence can be checked via truth tables. However, note that truth tables grow *exponentially*. Given a formula φ which contains the propositional variables $p_1, p_2, \ldots, p_n$, the truth table of φ has 2^n rows. Therefore, truth tables are not practical for large formulas.

Exercise 1.6

Check whether the formulas $\varphi := p \to q$ and $\psi := \neg p \vee q$ are semantically equivalent using a truth table.

Solution.

p	q	$p \to q$	$\neg p$	$\neg p \vee q$	$p \to q \leftrightarrow \neg p \vee q$
F	F	T	T	T	T
F	T	T	T	T	T
T	F	F	F	F	T
T	T	T	F	T	T

φ and ψ are equivalent since they evaluate to the same truth value for every entry in the truth table.

Example 1

Consider the formula $\varphi = a \wedge \neg(b \to c)$. Determine whether φ is satisfiable, valid, or both using its truth table.

Solution.

a	b	c	$b \to c$	$\neg(b \to c)$	φ
F	F	F	*T*	*F*	*F*
F	F	T	*T*	*F*	*F*
F	T	F	*F*	*T*	*F*
F	T	T	*T*	*F*	*F*
T	F	F	*T*	*F*	*F*
T	F	T	*T*	*F*	*F*
T	T	F	*F*	*T*	*T*
T	T	T	*T*	*F*	*F*

Figure 1.4: Truth table of $\varphi = a \wedge \neg(b \to c)$.

The formula is satisfiable but not valid since there is an entry that evaluates to *true*, but not all.

1.5 Encoding Example

To solve a problem using propositional logic, we need to find a proper encoding such that a satisfying assignment of the formula represents a solution for our problem. A satisfying assignment (if one exists) can be automatically found by an SAT solver. An SAT solver is a tool that takes as input a propositional formula and outputs either a satisfying assignment to the variables used in the formula if the formula can be satisfied.

Example 2

We use propositional logic to solve Sudoku. Rules: A Sudoku grid consists of a 9x9 square, which is partitioned into nine 3x3 squares. The goal is to write a number from 1 to 9 in each cell in such a way, that each row, each column, and each 3x3-square contains each number exactly once. Usually, several numbers are already given.

	6		7			1	5	
		3	9			8		
		2	3				4	9
		7			4			
	4			9			8	
			1			4		
6	7				9	3		
		9			2	5		
	2	8			7		6	

Sudoku

To model SUDOKU, we first need to define the propositional variables that we want to use in our formula.
One way to solve the problem is to define variables x_{ijk} for every row i, for every column j, and for every value k. This encoding yields to 729 variables ranging from x_{111} to x_{999}. Next, we need to define the constraints for the rows, the columns, the 3x3-squares and the predefined numbers.

- *Row-constraints: If* a cell in a row has a certain value, *then* no other cell in that row can have that value. For each i, and each k we have:

$$x_{i1k} \rightarrow \neg x_{i2k} \wedge \neg x_{i3k} \wedge ... \wedge \neg x_{i9k}$$

$$x_{i2k} \rightarrow \neg x_{i1k} \wedge \neg x_{i2k} \wedge ... \wedge \neg x_{i9k}$$

$$\vdots$$

$$x_{i9k} \rightarrow \neg x_{i1k} \wedge \neg x_{i2k} \wedge ... \wedge \neg x_{i8k}$$

- *Column-constraints: If* a cell in a column has a certain value, *then* no other cell in that column can have that value. For each j, and each k we have:

$$x_{1jk} \rightarrow \neg x_{2jk} \wedge \neg x_{3jk} \wedge ... \wedge \neg x_{9jk}$$

$$x_{2jk} \rightarrow \neg x_{1jk} \wedge \neg x_{2jk} \wedge ... \wedge \neg x_{9jk}$$

$$\vdots$$

$$x_{9jk} \rightarrow \neg x_{1jk} \wedge \neg x_{2jk} \wedge ... \wedge \neg x_{8jk}$$

- *Square-constraints: If* a cell in a 3x3 square has a certain value, *then* no other cell in that square can have that value. For the first square, we have for each k:

$$x_{11k} \rightarrow \neg x_{12k} \wedge \neg x_{13k} \wedge \neg x_{21k} \wedge \neg x_{22k} \wedge \neg x_{23k} \wedge \neg x_{31k} \wedge \neg x_{32k} \wedge \neg x_{33k}$$

$$\vdots$$

$$x_{33k} \rightarrow \neg x_{11k} \wedge \neg x_{12k} \wedge \neg x_{13k} \wedge \neg x_{21k} \wedge \neg x_{22k} \wedge \neg x_{23k} \wedge \neg x_{31k} \wedge \neg x_{32k}$$

  The constraints for the remaining squares are similar.

- *Predefined-number-constraints:* If a cell has a predefined value, we need to set the corresponding variable to true, e.g., the cell in the fifth row and the fifth column has the value 9. Therefore we have

$$x_{559}.$$

- *Cell-constraints:* Each cell must contain a number ranging from one to nine. For each i, and each j we have

$$x_{ij1} \vee x_{ij2} \vee ... \vee x_{ij9}.$$

  On its own, this constraint would allow for a cell to have more than one value. However, this is not possible due to the other constraints.

To construct the final propositional formula, all constraints need to be connected via conjunctions. A satisfying assignment for the final formula represents one possible solution for the Sudoku puzzle. In case that there does not exists a solution, the SAT solver would return UNSAT.

This chapter is based on
M. Huth and M. Ryan. *Logic in computer science : modelling and reasoning about systems.* Cambridge University Press, Cambridge [U.K.]; New York, 2004.

List of Definitions

Bibliography

[1] M. Huth and M. Ryan. *Logic in computer science : modelling and reasoning about systems*. Cambridge University Press, Cambridge [U.K.]; New York, 2004.