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

Graz University of Technology

Table of Contents

2

Propositional Logic

Why are we interested in logic as computer scientists? Our aim is to develop languages to formulate specifications and to model systems in such a way that we can *reason* about them *formally*.

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 are no taxis, Alice would be late to the appointment since the plane was late. Since she was not late, it must be the case that there were taxis at the airport. So the sentence after the 'therefore' *logically follows* from the sentences before it.

Example. If the sun is shining and John does not have his sunscreen with him, then he will get 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:

If p and not q, then r. Not r. p. Therefore, q.

By only considering the *logical structure* of the sentences instead of what the sentence really means, we are able to perform reasoning formally and automatically. Therefore, we need languages able to express sentences in such a way that brings out their logical structure. In this course, we will discuss the languages of propositional logic and predicate logic. We will begin with the language of propositional logic.

2.1 Declarative Sentences

Propositional logic, like most branches of logic, is only concerned with sentences that make statements, so called *declarative sentences*, or *propositions*. Declarative sentences state a fact, or describe a state of affairs. A declarative sentence can be declared as *true* or *false*. It must be one or the other. (It can't be both, nor can it be undefined). True and false are referred to as *truth values*.

Example. Examples for declarative sentences are:

- 10 divided by 5 is 3.
- Mozart was born in Austria.
- Santa Clause lives at the North Pole.
- It's raining in Graz right now.
- If it rains the street is wet.

All of these sentences can be determined to be either true or false. The first sentence can easily be demonstrated to be false, since it holds that "10 divided by 5 is 2". The second and third sentence need to be examined more closely. You need to know who Mozart and Santa Clause are and you need to gather evidence about them to find out whether or not those statements are true. The fourth sentence has no absolute truth value, but at any given point in time it is either true or false.

Non-declarative sentences are, amongst others, questions, orders, and greetings.

Example. Examples for non-declerative sentences are:

- Do you like ice cream?
- Take the dog for a walk!
- Hello!
- May the force be with you!

2.2 Syntax of Propositional Logic

Propositional logic formulas consist of *atomic propositions*, *logical operators*, and *parentheses*.

Atomic Propositions

Atomic propositions are declarative sentences, or parts of declarative sentences, that can be assigned a truth value and cannot be split up into two or more sentences that can be assigned a truth value themselves. For brevity, atomic propositions are shortened to a single lower case letter.

An atomic proposition can be represented by a *propositional symbols*, or *propositional variable* such as p, p_1 or q.

Example.

p : It is Sunday.
q : Students have to present their solutions at the exercise classes.

These statements can be true or false and cannot be shortened any further, so they are atomic propositions.

Logical Operators

We form compound statements from atomic statements, by joining them using connectives.

Example. As an example, consider the sentence:

John is happy *and* he his hungry.

This is a compound statement: the word 'and' is an example of a connective. The atomic propositions are p : "John is happy" and p : "John is hungry".

The symbols $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\rightarrow$ (implies), and $\leftrightarrow$ (equivalent) are used to represent the connectives and are called the *logical operators*.

Example. Given the sentence:

"Two straight lines can either be parallel or intersecting, but not both."

To model this sentence as a formula, we define the atomic propositions $p :=$ "lines are parallel", and $q :=$ "lines are intersecting". Then, the sentence can be modelled by the formula $\varphi = (p \vee q) \wedge \neg(p \wedge q)$.

Example. Given the sentence:

"Tomorrow is Tuesday, if and only if today is Monday."

To model this sentence as a formula, we define the atomic propositions $m :=$ "today is Monday", and $t :=$ "tomorrow is Tuesday". Then, the sentence can be modelled by the formula $\varphi = (m \leftrightarrow t) \equiv (m \rightarrow t) \wedge (t \rightarrow m)$.

Example. The atomic propositions are defined by $a :=$ "k is even", $b :=$ "k is greater than seven", and $c :=$ "k is prime". Using the defined atomic propositions, we can formally model the following sentences:

- "Number k is even or greater than 7" is modelled by $a \vee b$
- "If k is both even and prime, then in particular it is even" is modelled by $a \wedge c \rightarrow a$
- "Either k is even, or k is prime, and not both" is modelled by $(a \vee c) \wedge \neg(a \wedge c)$

Definition of the Syntax of Propositional Logic

Formulas in propositional logic are strings over the alphabet of atomic propositions, logical operators and parentheses. However, the string $pq()(\wedge)$ does not make a lot of of sense as far as propositional logic is concerned. We have to define those strings which we want to call formulas. We call such formulas well-formed.

Definition. The *well-formed formulas* of propositional logic are those which we obtain by using the construction rules below, and only those, finitely many times:

- Every propositional symbol is a well-formed formula. We use p, p_1, $p_2 \ldots$, q, q_1, $q_2 \ldots$ as propositional symbols.
- For any well-formed formula φ, $\neg\varphi$ is also a well-formed formula.
- For any two well-formed formulas φ and ψ, the following are also well-formed formulas: $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$ and $\varphi \leftrightarrow \psi$.
- Any well-formed formula φ may be enclosed in parentheses and will still be a well-formed formula, i.e., (φ).

Note: No other string of symbols is a propositional formula.

The definition of well-formed formulas in propositional logic can also be defined using a grammar in Backus-Naur form (BNF). In that form, the above definition reads more compactly as:

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

Parentheses and Operator Precedence

Parentheses determine evaluation order. The operator precedence define how strongly an operator is binding to its sub-formulas. Precedence rules can be given:

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

E.g., $p \wedge q \vee r \rightarrow p$ evaluates as $((p \wedge q) \vee r) \rightarrow p$. Associativity rules can be given:

$$\text{Associate to the left: } \vee \ \wedge$$

$$\text{Associate to the right: } \rightarrow \ \leftrightarrow$$

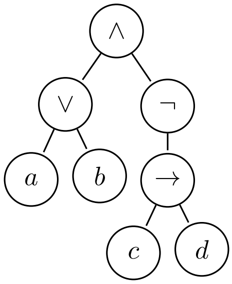
Example. The formula $p \wedge q \wedge r \rightarrow p \rightarrow r$ evaluates as $((p \wedge q) \wedge r) \rightarrow (p \rightarrow r)$.

The operator precedence is very useful in reducing the number of parentheses. For example, the $\neg$ operator has highest precedence. So, e.g., $\neg p \wedge q$ evaluates as $(\neg p) \wedge q$, and, for the other evaluation order, you would write $\neg(p \wedge q)$.

Parse Trees

How can we show that a string is a well-formed formula? Consider the following formula: $\varphi = (a \vee b) \wedge (\neg(c \rightarrow d))$.

The easiest way to verify whether some formula φ is a well-formed formula is by trying to draw its parse tree. The parse tree for φ is given below.



At its top level, the formula is a conjunction. The left sub-formula is a disjunction and the top level connective of the right sub-formula is first a negation followed by an implication.

To verify if φ is well-formed, we need to check whether all leaves are atomic variables. All other nodes are labeled with logical operators. In the case of $\neg$ there is only one subtree coming out of that node. In the cases $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$ we must have two subtrees.

When drawing the parse tree do not forget about the operator precedence to decide on the current top level connective.

2.3 Semantics of Propositional Logic

So far, we have discussed propositional logical formulas. We noted that formulas derive from declarative sentences, or statements. We would now like to determine whether a formula is *true* or *false* based on the truth values of the statements that comprise the formula and the meaning of the logical operators. Truth is a *semantic* notion, in that it ascribes a type of *meaning* to certain formulas. There are different notations for truth values, see Table 2.1.

true	**false**
T	F
1	0
$\top$	$\bot$

Figure 2.1: Different notations of truth values.

The semantics of propositional logic are the rules we give for our propositional connectives. They are defined in the *truth tables* 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 2.2: Truth tables for $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$.

φ	$\neg\varphi$
F	T
T	F

Figure 2.3: Truth table for negation.

How can you check that a formula φ and a formula ψ are semantically equivalent? For example, to understand the meaning (= the semantics) of an implication, consider the formula $\varphi = p \rightarrow q$ and the formula $\psi = \neg p \vee q$. By using the truth tables for $\neg$ and $\vee$ you can check that $p \rightarrow q$ evaluates to T if and only if $\neg p \vee q$ does so. Since this is the case, this means that φ and ψ are are semantically equivalent.

What is the caveat in using truth tables? Given a formula φ which contains the propositional atoms $p_1, p_2, \ldots, p_n$. The truth table of φ has 2^n rows. Each row listing one of the possible combinations of truth values for $p_1, p_2, \ldots, p_n$. Since the number of rows grows exponentially, truth tables are not practical for large formulas.

Models

Models are also called *valuations*, *interpretations*, or *assignments*.

Definition. A *model* $\mathcal{M}$ of a propositional formula φ is an assignment of each propositional variable in φ to a truth value.

Example. Consider the formula $\varphi = \neg p \wedge q$. The map which assigns $\bot$ to p and $\top$ to q is a model $\mathcal{M}$ for φ, i.e,: $\mathcal{M} = \{p = \bot, q = \top\}$.

Furthermore, we distinguish between:

- *Satisfying Model:* truth assignment such that the formula resolves to *true*.
- *Falsifying Model:* truth assignment such that the formula resolves to *false*.

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

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

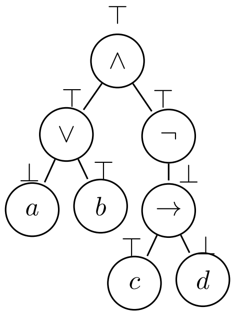
We call $\models$ the *semantic entailment* relation.

Evaluation of a Logical Formula under a given Model

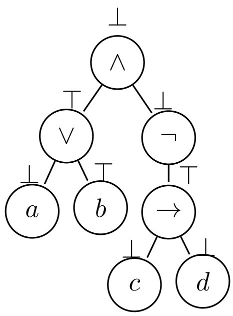
How can we determine the truth value of a formula φ for a given model $\mathcal{M}$? Given the meaning of the atomic propositions defined by $\mathcal{M}$, we can use the parse tree of φ to propagate the truth values upwards. The associated truth value of the root is the truth value of φ.

Example. Let $\varphi = (a \vee b) \wedge (\neg(c \rightarrow d))$ and $\mathcal{M}_1 = \{a = \bot,\ b = \top,\ c = \top,\ d = \bot\}$.

The parse tree if given below. We assign the truth values of the atomic propositions to the lowest level. Then the truth values are propagated upwards, e.g., since $\bot \vee \top = \top$, the left sub-tree evaluates to true. $\top \rightarrow \bot$ evaluates to false, etc. Finally, to evaluate the meaning of φ, we compute $\top \wedge \top$ which is $\top$. Formally, we have $\varphi^{\mathcal{M}_1} = \top$, or $\mathcal{M}_1 \models \varphi$.



Example. Let us evaluate the same formula $\varphi = (a \vee b) \wedge (\neg(c \rightarrow d))$ under a different model $\mathcal{M}_2 = \{a = \bot,\ b = \top,\ c = \bot,\ d = \bot\}$. As illustrated below, the formula now evaluates to false. Formally, we have $\varphi^{\mathcal{M}_2} = \bot$, or $\mathcal{M}_2 \not\models \varphi$.



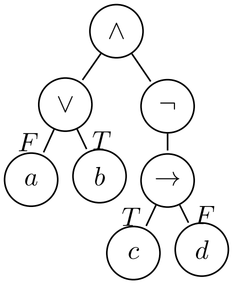
Example: If the truth values for the formula

$$(a \vee b) \wedge (\neg(c \to d))$$

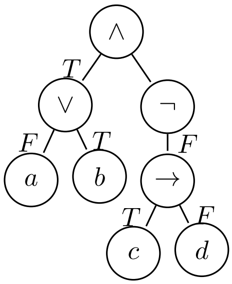
are

$$a = F,\ b = T,\ c = T,\ d = F$$

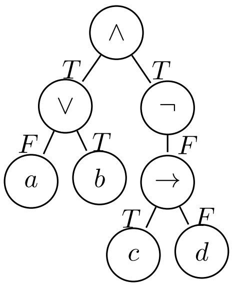
and the truth values are assigned to the resulting parse tree



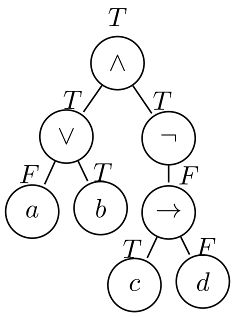
then the sections of the formula can be evaluated bottom up. In the case at hand $F \vee T$ evaluates to true, $T \to F$ evaluates to false, resulting in the parse tree



which can be further evaluated. $\neg F$ evaluates to true, resulting in the parse tree



Now only the section $T \wedge T$ remains. It evaluates to true, resulting in the final parse tree



Since the top most node of the parse tree evaluates to true the formula is true.

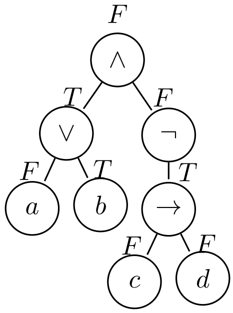
If the truth values for the same formula

$$(a \vee b) \wedge (\neg(c \rightarrow d))$$

are

$$a = F,\ b = T,\ c = F,\ d = F$$

the formula evaluates to false, as can be seen in the parse tree



2.4 Semantic Entailment, Equivalence, Satisfiability and Validity

Next, we discuss elementary concepts of semantics in propositional logic: the concepts of semantic entailment, semantic equivalence, validity, and satisfiability.

Semantic Entailment

We say that a formula φ semantically entails a formula ψ, if any model that satisfies φ also satisfies ψ, i.e., we have $\varphi \models \psi$ if $\mathcal{M} \models \varphi$ implies $\mathcal{M} \models \psi$.

Definition. Let φ and ψ be formulas in propositional logic. We say that $\varphi \models \psi$ if and only if every model $\mathcal{M}$ that satisfies φ ($\mathcal{M} \models \varphi$) also satisfies ψ ($\mathcal{M} \models \psi$).

Example. It holds that $a \models a \vee b$, $a \vee b \not\models a$, $(p \wedge q) \models p$, and $(p \vee q) \not\models p$.

Example. It holds that $\perp \models \varphi$. The definition of semantic entailment requires that all models that satisfy the entailing formula also satisfy the entailed formula. Since a contradiction does not have any satisfying models, a contradiction entails any possible formula.

Figure 2.4 gives a visual representation of the semantic entailment relation. In the illustration we have the following area encodings.

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

Semantic Equivalence

Two formulas φ and ψ are called *semantically equivalent* if both evaluate to true under the same models.

Definition. Let φ and ψ be formulas in propositional logic. We say that φ and ψ are semantically equivalent if and only if $\varphi \models \psi$ and $\psi \models \varphi$ holds.

In that case we write $\varphi \equiv \psi$.

Example. $a \rightarrow b \equiv \neg b \rightarrow \neg a$ and $a \rightarrow b \equiv \neg a \vee b$.

Validity

A formula is called *valid*, if the formula evaluates to *true* under all possible models.

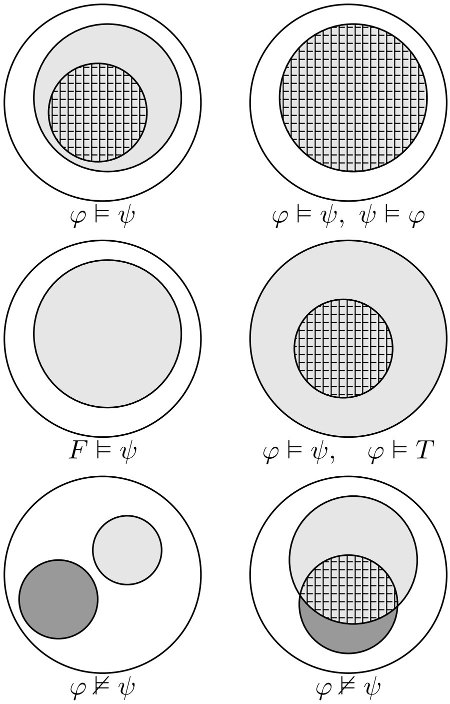


Figure 2.4: Visualisation of the semantic entailment relation.

Definition. Let φ be a formula of propositional logic. We call φ valid if $\vDash \varphi$ holds.

A valid formula is also called a *tautology.*

Example. The formula $\varphi = (p \vee \neg p)$ is valid.

Example. It holds that $\psi \vDash \top$. All possible models satisfy a tautology. Thus, any formula entails true.

Satisfiability

One of the most important questions in logic is the question whether there exists a model under which the formula evaluates to true.

Definition. Given a formula φ in propositional logic, we say that φ is *satisfiable* if it has a model in which is evaluates to *true.*

Example. Let $\varphi = (p \vee \neg q) \rightarrow p$. φ is satisfiable since it evaluated to *true* under the model $\mathcal{M} = \{p = \top, q = \top\}$. Note, that φ is not valid, since it evaluates to *false* under $\mathcal{M}' = \{p = \bot, q = \bot\}$.

A formula that is *not* satisfiable is called a *contradiction*, i.e., the formula evaluates to *false* under all possible models.

Example. The formula $\varphi = (p \wedge \neg p)$ is not satisfiable.

A simple way of checking satisfiablity or validity for small formulas is using its truth table.

Example. Consider the formula $\varphi = a \wedge \neg(b \to c)$. Its truth table is shown in Figure 2.5. The formula is satisfiable since there is an entry at the table that is *true*. The formula is not valid since there are entries at the table that are *false*.

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

2.5 Encoding Example

In order to solve a problem using propositional logic, we need to find a proper encoding in propositional logic 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 a SAT solver. A 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 or UNSAT if it can not.

Example. 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 one 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

In order 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:

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

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

$$\vdots$$

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

- *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:

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

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

$$\vdots$$

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

- *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

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

  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 sovler would return UNSAT.

Declaration of Sources

This chapter was based on the following book.

Michael Huth, Mark Dermot Ryan: *Logic in Computer Science: Modelling and Reasoning about Systems.* June 2004. Cambridge University Press. ISBN:978-0-521-54310-1