Lecture Notes on Propositional and Predicate Logic

Martin Pilát

Based on lecture by Petr Gregor

OCTOBER 17, 2017

Introduction

Generally, logic is a study of arguments and inferences. While it started as philosophical discipline in ancient times, it is now widely studied in mathematics and computer science. As such, logic provides the basic language and tools for most of mathematics. It studies different prove systems and discusses whether they are sound (everything they prove is valid) and complete (everything that is valid can be proven).

While logic provides rather low-level tools for mathematics and computer science, it still has rather wide applications. For example, in some areas of artificial intelligence, logic is used to represent the knowledge of the intelligent agents and reason about it. The agents than use logical reasoning (theorem proving) to decide what to do next, or prove that a certain action is safe in a given environment. Another important area is formal software verification, where logic (and, again, theorem proving) can be used to formally verify, that a program indeed does what it should according to a specification. This is essential while implementing e.g. cryptographic protocols. Formal verification is also used while designing digital circuits.

There are also attempts to formalize the whole mathematics in logic and use computers to check that all the proofs are correct. For example, Mizar[1] is a system that aims to re-create most of the mathematics with formal and verified proofs. The verification starts from the basic mathematical axioms – in the case of the Mizar system, authors of so called Mizar articles can use only axioms of set-theory and theorems from previously verified articles. Therefore, everything published in the Mizar mathematical library is verified to be a correct consequence of the base axioms.

[1] http://www.mizar.org/library/

Logic serves as the formal language of mathematics, and therefore logic also needs to formally specify the syntax of the language. The syntax defines, what is a valid logical formula and what is not, however the meaning and validity of a formula is given by the semantics of the language. Logic itself prescribes the meaning of only a handful of symbols – namely the logical connectives ($\wedge, \vee, \rightarrow, \leftrightarrow, \neg$) and the quantifiers ($\forall, \exists$). Additionally, in languages with equality, the meaning of "$=$" is also given. All other symbols used in logical formulas can have arbitrary meaning, which is given by the semantics. So, for example, if we write a formula $(\forall x)(\forall y)(x + y = y + x)$, we cannot discuss its validity before defining the meaning of "$+$". The formula is valid if we are talking about the real numbers and "$+$" denotes their addition, however, the symbol "$+$" can also represent (quite un-

usually) the multiplication of square matrices, and in such a case, the formula is not valid.

There are different levels of the language of logic. In propositional logic, only propositional variables (those that are either true or false) and the logical connectives can be used. In first order logic, we can additionally use functions, relations and quantifiers for variables that range objects from some universe. In second order logic, there are additionally quantifiers for sets of objects in the universe (and, more specifically, for functions and relations). In higher order logic, we also have variables for sets of sets of objects. For example, a formula in propositional logic

$$(d \wedge c) \rightarrow s$$

can express that if it is dark and clear outside, the stars are visible. In a first order language, we can have a more complex formula

$$(\forall x)(\forall y)(S(x) \wedge E(y) \rightarrow (L(x,y) \rightarrow P(x,y)))$$

that expresses that if x is a student ($S(x)$) and y is an exam ($E(y)$), if x learns for y ($L(x,y)$) then x passes y ($P(x,y)$). As an example of second order language, we can write the axiom of induction:

$$(\forall P)(P(0) \wedge (\forall n)(P(n) \rightarrow P(n+1)) \rightarrow (\forall n)P(n))\,.$$

In the lecture, we will deal mostly with propositional and first-order logic, however there are also other extension of logic. For example, in multi-agent systems, so called modal logic is often used to represent the knowledge. In modal logic, there are special modalities, that can further qualify a statement. For example, there is a modality that says that a statement may be true, or must be true. Other types of modal logic contain modalities that express knowledge of other agents (e.g. "agent A knows that statement S is true" can be written as $\mathrm{K}_A.S$, or even "agent A knows that agent B knows that statement S is true" ($\mathrm{K}_A.\mathrm{K}_B.S$)). Another interesting type of modal logic is temporal logic, which contain modalities about time and can express e.g. "statement S will be true sometimes in the future".

About these lecture notes

In these lecture notes, logic is presented for students of computer science. Therefore, focus is given to areas most needed for computer scientists. For example, we use the more intuitive tableau method instead of the Hilbert-style prove systems. We also explain the resolution method in logic as a background to Prolog and logical programming. The more advanced topics on decidability and incompleteness are explained in a more informal way.

You are currently reading the first version of the lecture notes, which can, and most probably will, contain some errors. If you find an error, or if something is not clear, do not hesitate to contact the author by e-mail[2], or, alternatively, create an issue in the GitHub repository of the book[3].

[2] Martin.Pilat@mff.cuni.cz

[3] https://github.com/martinpilat/logic-book

There are also other resources you may want to check. One of them are the presentations created by Petr Gregor for his version of the lecture[4], that serve as a base for these lecture notes.

[4] http://ktiml.mff.cuni.cz/~gregor/logics/index.html

Informal! The author of these notes sometimes likes to explain things in a more intuitive way with some not-so-formal examples and metaphors. While he believes these can help to get better understanding of the given concept, they sometimes (read "often") are rather informal and have some limitations. Therefore, in these notes, they will be set in boxes like the one you are reading just now with a bold "**Informal!**" warning. The information contained in these boxes is always non-essential to the rest of the text and can be (some would even argue that should be) skipped.

Preliminaries

Like many mathematical texts, these lecture notes also assume that the reader has some basic knowledge. The most important concepts (many of which should sound familiar) are briefly introduced in this short section, both to provide a single place where these can be found and to introduce the notation used in these lecture notes.

We will start with the basic set-theoretic notions. The most basic of these is the *class*. Each property of sets $\varphi(x)$ defines a class $\{x|\varphi(x)\}$. Some classes are also sets, those that are not are called *proper classes*. The distinction between sets and classes is probably new for most of the readers. Why would we need any other collections of objects than sets? How is it possible, that there is a collection of objects, which is not a set? The reason to distinguish between these to is that if everything was considered a set, we could find paradoxes in the set theory. For example, if we had a set of all sets that do not contain themselves, does this set contain itself, or not? Let us assume, it does, but then, by definition, it does not. If we instead assume it does not contain itself, then, again, by definition, it does. This so called Russel's paradox can be avoided by using a notion of classes, that cannot contain other classes.

Informal! A class can be understood as any collection of sets that can be described by the language of set theory. However, some of these collections do not make much sense and can lead to paradoxes. Therefore, any collection, that would lead to some paradox is denoted as a proper class instead of a set and the paradoxes can thus be avoided.

The other set-theoretic notions should be much more familiar. We use $x \in y$ to denote that x is a member of set y, $x \notin y$ and $x \neq y$ are shortcuts for $\neg(x \in y)$ and $\neg(x = y)$. A set containing exactly elements $x_0, x_1, \ldots, x_n$ is denoted as $\{x_0, x_1, \ldots, x_n\}$. A set with only one element $\{x\}$ is called a *singleton* and a set with two elements $\{x_0, x_1\}$ is called an *unordered pair*. We will also use the common notation for set operations: $\emptyset$ denotes an *empty set*, $\cup$ and $\cap$ denote *union* and *intersection* of sets. The $\setminus$ is the *set difference* operator and $\triangle$

is the *symetric set difference* operator

$$x \triangle y = (x \setminus y) \cup (y \setminus x).$$

Two sets are *disjoint*, if their intersection is an empty set, and $x \subseteq y$ denotes that x is a subset of y (all elements of x are also elements of y). The set of all subsets of a set x – the *power set of* x – is denoted as $\mathcal{P}(x)$. The *union of set* x, $\bigcup x$, is the union of all sets contained in x. A *cover of a set* x is a set $y \subseteq \mathcal{P}(x) \setminus \emptyset$, such that $\bigcup y = x$, if all the sets in the cover y are mutually disjoint, than y is a *partition of* x.

The definition of an unordered pair can be used to define the *ordered pair* $(a,b) = \{a,\{a,b\}\}$ and an *ordered n-tuple* $(x_0,\ldots,x_{n-1}) = ((x_0,\ldots,x_{n-2}),x_{n-1})$ for $n > 2$. A Cartesian product of two sets a and b is $a \times b = \{(x,y)|x \in a, y \in b\}$ and the Cartesian power of a set x is $x^0 = \{\emptyset\}$, $x^n = x^{n-1} \times x$. A *binary relation* R is a set of ordered pairs. The *domain of* R is defined as $\mathrm{dom}(R) = \{x|(\exists y)(x,y) \in R\}$, the *range of* R is similarly $\mathrm{rng}(R) = \{y|(\exists x)(x,y) \in R\}$. The *extension of* x *in* R is the set $R[x] = \{y|(x,y) \in R\}$. The symbol R^{-1} denotes the *inverse relation* $R^{-1} = \{(y,x)|(x,y) \in R\}$. The *restriction of* R *to a set* z is defined as $R \restriction z = \{(x,y) \in R|x \in z\}$. Two relations can also be *composed* into one, $R \circ S = \{(x,z)|(\exists y)((x,y) \in R \wedge (y,z) \in S\}$. The *identity relation* on set z, $\mathrm{Id}_z = \{(x,x)|x \in z\}$.

A binary function f is a special type of binary relation where for every $x \in dom(f)$ there is exactly one y such that $(x,y) \in f$, then, y is the value of f in x denoted as $f(x)$. $f : X \to Y$ denotes a function f with $\mathrm{dom}(f) = X$ and $\mathrm{rng}(f) \subseteq Y$. The set of all such functions is ${}^{Y}X$. A function $f : X \to Y$ is a *surjection* (onto) if $\mathrm{rng}(f) = Y$, and it is an *injection* (one-to-one) if for any $x,y \in dom(f)$, $x \neq y \to f(x) \neq f(y)$. A function that is both a surjection and injection is called a *bijection*. Similarly to relation, we can define the inverse function f^{-1}, and the composition of functions $f : X \to Y$ and $g : Y \to Z$ as a function $f \circ g$ with $(f \circ g)(x) = g(f(x))$. The image of a set A, denoted as $f[A]$ is the set of function values for all elements of A, $f[A] = \{y|(x,y) \in f, x \in A\}$.

There are also two special types of relations which will be important later: equivalences and orders. An equivalence on a set X is relation that is reflexive ($R(x,x)$ for all $x \in X$), symmetric ($R(x,y) \to R(y,x)$ for $x,y \in X$) and transitive ($(R(x,y) \wedge R(y,z)) \to R(x,z)$) for all $x,y,z \in X$). The extension of x in R is called the equivalence class of x and is also denoted as $[x]_R$. $X/R = \{R[x]|x \in X\}$ is the *quotient set of* X *by* R. The quotient set is always a partition of X and every partition of X also defines an equivalence on X (two elements are equivalent if they are in the same set in the partition).

The other important types of the relations are the orders, usually an order is denoted as $\leq$. Such a relation is a *partial order of a set* X, if it is reflexive ($x \leq x$ for $x \in X$), antisymmetric ($x \leq y \wedge y \leq x \to x = y$ for $x,y \in X$) and transitive ($x \leq y \wedge y \leq z \to x \leq z$ for $x,y,z \in X$). If, additionally, for every two elements $x,y \in X$ it holds that $x \leq y$ or $y \leq x$ (dichotomy) than $\leq$ is a total (linear) order. It is a well-order if additionally every non-empty subset of Xhas a least element. Finally,

and order of X is dense, if X is not a singleton and for every two elements $x, y \in X$, there is another element $z \in X$ between these two $(x < y \rightarrow (\exists z)(x < z \wedge z < y))$, where $a < b$ means that $a \leq b \wedge a \neq b$.

For example, the common ordering of natural numbers ($\leq$ on $\mathbb{N}$) is a linear well-order (as every two natural numbers are comparable and every subset of natural numbers has a least element under this order), however, it is not a dense order, as for example there is no natural number between 0 and 1. On the other hand, the common ordering of rational numbers is a dense linear order (there is a rational number between any pair of distinct rational numbers), however it is not a well-order, as e.g. the set $\{x \in \mathbb{Q} \,|x \leq 0\}$ has no least element.

The natural numbers can be defined using the empty set in an inductive way – $0 = \emptyset, 1 = \{0\} = \{\emptyset\}, 2 = \{0, 1\} = \{\emptyset, \{\emptyset\}\}, \ldots, n = \{0, \ldots, n-1\}, \ldots$. The set of all natural numbers $\mathbb{N}$ is the smallest set containing $\emptyset$ and closed under the operation of successor $S(x) = x \cup \{x\}$. The other common sets of numbers are the integers, which can be defined as the $\mathbb{Z} = (\mathbb{N} \times \mathbb{N})/\sim$, with $(a, b) \sim (c, d)$ if and only if $a + d = b + c$. Similarly, the set of rational numbers $\mathbb{Q}$ can be defined as $\mathbb{Q} = (\mathbb{Z} \times (\mathbb{Z} \setminus \{0\}))/\sim$, with $(a, b) \sim (c, d)$ if and only if $ad = bc$. The definition of real numbers $\mathbb{R}$ is more complex. These are usually defined as cuts of the rational numbers $\mathbb{Q}$, where a cut is a partition of $\mathbb{Q}$ into two sets A and B, where all numbers in B are greater than all numbers of A, and A has no greatest element. For example, the cut corresponding to the irrational number $\sqrt{2}$ is $A = \{a \in \mathbb{Q} | a^2 < 2 \vee a < 0\}, B = \{b \in \mathbb{Q} | b^2 > 2 \wedge b > 0\}$.

Another important notion for the rest of the lecture deals with the cardinality ("size") of sets. A set X has a cardinality smaller or equal to set Y ($X \preceq Y$) if there is an injective function $f : X \rightarrow Y$. If there is a bijection $f : X \rightarrow Y$ than we say that X and Y have the same cardinality ($X \approx Y$), finally X has strictly smaller cardinality than Y ($X \prec Y$) if ($X \preceq Y \wedge \neg(X \approx Y)$). For each set x, there is a cardinal number $\kappa \approx x$, denoted as $|x| = \kappa$. A set X is *finite* if $|X| = n$ for some $n \in \mathbb{N}$. It is *countable*, if its finite or if $|x| = |\mathbb{N}| = \omega$. Otherwise, it is *uncountable*. The cardinality of $\mathcal{P}(\mathbb{N})$ is called the continuum.

It is interesting to know the cardinality of the common sets of numbers. Obviously, the set of natural numbers $\mathbb{N}$ is countable. A less obvious fact is that the sets of integers and rational numbers also have the same cardinality and are therefore also countable. For the integers, we can create an infinite sequence of integers $s = \langle 0, 1, -1, 2, -2, 3, -3, \ldots \rangle$, then a function $f(i) = s_i$ is an injective function $\mathbb{N} \rightarrow \mathbb{Z}$, therefore $\mathbb{Z} \preceq \mathbb{N}$. The other inequality ($\mathbb{N} \preceq \mathbb{Z}$) is obvious (use identity as the injective function). In order to show that the set of rational numbers $\mathbb{Q}$ is also countable, we can create a function $f(\frac{p}{q}) = 2^{|p|} 3^q 5^{\text{sign}(p)}$ (we consider only cases where $p \in \mathbb{Z}, q \in \mathbb{N} \setminus \{0\}$, which clearly covers all the rationals), this is again an injective mapping $\mathbb{Q} \rightarrow \mathbb{N}$ and therefore $\mathbb{Q} \preceq \mathbb{N}$. As before, the other inequality is trivial. Finally, we can show that the set of real numbers $\mathbb{R}$ has bigger cardinality than the set of natural numbers $\mathbb{N}$. Obviously $\mathbb{N} \preceq \mathbb{R}$ as $\mathbb{N} \subseteq \mathbb{R}$. Let us assume both the sets have the

same cardinality, in such a case there is bijection $f : \mathbb{N} \to \mathbb{R}$. We will now define a new real number r in the following way. The integer part of the number is 0, the first digit after the decimal point is different from the first digit after decimal point in $f(0)$, the second digit is different from the second digit in $f(1)$, and so on[5]. This real number is different from all the numbers in $\{f(0), f(1), \dots\}$ as it differs from the number $f(i)$ in the i-th digit after the decimal point. This is a contradiction with the assumption that f is a bijection between $\mathbb{N}$ and $\mathbb{R}$ and therefore $\mathbb{N} \prec \mathbb{R}$.

[5] If we write the decimal value of the number r as $r = 0.r_0r_1r_2\dots$, where r_i is the i-th decimal digit, we can define $r_i = (f(i)_i + 1) \bmod 10$, where $f(i)_i$ is the i-th decimal digit of $f(i)$.

We will conclude the discussion of cardinalities by showing the Cantor's theorem.

Theorem 1 (Cantor)**.** *For every set x, $x \prec \mathcal{P}(x)$.*

Proof. First, $f(x) = \{x\}$ is an injection $X \to \mathcal{P}(x)$ and therefore $x \preceq \mathcal{P}(x)$. Suppose there is also an injective $g : \mathcal{P}(x) \to x$. We can define a set $y = \{g(z) | z \subseteq x \wedge g(z) \notin z\}$. Now, similarly to the Russel's paradox, $g(y) \in y$ if and only if $g(y) \notin y$, which is a contradiction, and therefore there cannot be any such injective g and so $x \prec \mathcal{P}(x)$. □

As the tableau method used in this lecture relies on trees, we will conclude this preliminary section by a brief discussion on trees. Most of the readers are probably familiar with finite trees, however, we will sometimes need to work with infinite trees and therefore we define a *tree* as a set T with a partial order $<_T$ (called the tree order) with a unique least element (*the root*) and in which the set of predecessors of any element is well-ordered by $<_T$. In this definition a branch is a maximal linearly ordered subset of T. Apart from this difference in definition, we will use the common terminology on trees from the graph theory. For simplicity, we will only consider finitely branching trees, where each node except the root has an immediate predecessor[6]. In such trees we can define the *levels of the tree*. The root is on the level 0, the sons of the nodes on the $(n-1)$-th level are on level n. The depth of tree is maximal $n \in \mathbb{N}$ of a non-empty level. In case the tree has an infinite branch it has an infinite depth ω. In an n-ary tree, each node has at most n sons and a tree is finitely branching if each node has a finite number of sons.

[6] This means, we will not deal, for example, with trees where the nodes would be set of rational numbers $\mathbb{Q}$ and the tree order $<_T$ would be the common order on $\mathbb{Q}$.

Lemma 1 (König)**.** *Every infinite, finitely branching tree contains an infinite branch.*

Proof. The root of the tree has only finitely many sons, therefore there is a son of the root that is infinite. We choose this son and continue in the same way with his sons, thus constructing an infinite branch. □

Apart from the tree order $<_T$ we sometimes need to work with *ordered trees* where the sons of each node are additionally ordered from left to right with a *left-to-right order* $<_L$. In a *labeled tree* each node also contains an additional information. For example, the formula

$$(p \wedge q) \to q$$

can be represented as the labeled ordered tree on the left.

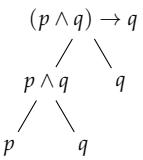

Figure 1: The labeled ordered tree representing the formula $(p \wedge q) \to q$.

Part I

Propositional Logic

Propositional Formulas and Models

In this chapter, we start the discussion of propositional logic. We will define, how propositional formulas look, what is a model in propositional logic and we will also discuss some special forms of formulas.

Propositional logic is the more basic type of logic (and predicate logic is an extension of propositional logic in a sense). Propositional formulas (propositions) are created from so called *propositional variables* that represent an atomic fact which can either be true or false. These propositional variables can only be connected by common logic connectives ($\rightarrow$, $\leftrightarrow$, $\wedge$, $\vee$, $\neg$). Logical formulas can additionally use parentheses to indicate the order of application of connectives. While the propositional formulas are simple compared to formulas in other types of logic, they are still useful. One of the most important problems in propositional logic and in computer science in general is the satisfiability of propositional formulas (SAT). Many other NP-complete problems are often solved by transformation to the SAT problem and using one of the existing SAT solvers.

Syntax of Propositional Logic

The set of propositional variables is often called $\mathbb{P}$ and the variables themselves are usually named p, q, r, s or $p_0, p_1, \ldots, q_0, q_1$, or similarly. Now, we can formally define the propositional formula (over $\mathbb{P}$).

Definition 1. Let $\mathbb{P}$ is the set of propositional variables, than

1. Every propositional variable from $\mathbb{P}$ is a propositional formula.
2. If φ and ψ are propositional formulas, than $(\varphi \rightarrow \psi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \leftrightarrow \psi)$, and $(\neg\varphi)$ are propositional formulas.
3. Every propositional formula is created by finite application of the two rules above.

The last part of the definition ensures that every formula is finite, this also means that each formula can contain only a finite number of distinct variables. The set of propositional variables used in a formula φ will be denoted as $\mathrm{var}(\varphi)$. On the other hand, the set of all propositional formulas using only variables from a set $\mathbb{P}$ will be denoted as $\mathrm{VF}_{\mathbb{P}}$.

Formulas are thus strings created from propositional variables, logical connectives, and parentheses, that fulfill the conditions in the

definition above. A substring of such a string that also fulfills the conditions is called a *sub-formula*.

The formal definition of formula dictates the use of parentheses around every sub-formula, which can be rather cumbersome. Therefore, we define priorities of the logical connectives and can thus omit some of the parentheses. The standard priorities are such, that the negation ($\neg$) has the highest priority (therefore parentheses around $(\neg\varphi)$ can always be omitted), conjunction and disjunction ($\vee, \wedge$) have "middle" priority, and implication and equivalence ($\rightarrow, \leftrightarrow$) have the lowest priority. Therefore, we can write $\varphi \wedge \psi \rightarrow \neg\varphi \vee \xi$ instead of $((\varphi \wedge \psi) \rightarrow ((\neg\varphi) \vee \xi))$.

Each formula can be also represented by a so called *formation tree*, which is a finite ordered tree, whose nodes are labeled with propositions – the leaves are labeled with propositional variables, if a node has label $(\neg\varphi)$, it has a single son labeled with φ, and if a node has label $(\varphi \rightarrow \psi), (\varphi \wedge \psi), (\varphi \vee \psi)$, or $(\varphi \leftrightarrow \psi)$, it has two sons, the left one has label φ, and the right one has label ψ. For example, a formula $p \wedge q \rightarrow \neg(p \vee s)$ is represented by the formation tree on the left.

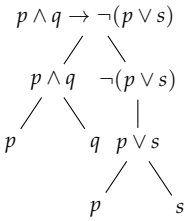

Figure 2: The formation tree representing the formula $p \wedge q \rightarrow \neg(p \vee s)$.

It is simple to show (by the induction on the number of nested parentheses) that each formula is associated with a unique formation tree.

Semantics of Propositional Logic

Once we have the formal definition of the formula (the syntax of propositional logic), we can define its semantics (what the formula means). The propositional variables represent atomic statements, that can have one of two truth values – either 0 (false) or 1 (true). The truth value of the whole proposition in then given by the truth values of the variables and by the semantics of the logical connectives, which is given in Table I bellow.

p	q	$\neg p$	$p \vee q$	$p \wedge q$	$p \rightarrow q$	$p \leftrightarrow q$
0	0	1	0	0	1	1
0	1	1	1	0	1	0
1	0	0	1	0	0	0
1	1	0	1	1	1	1

Table 1: The semantics of logical connectives

We can also consider the table above a definition of Boolean functions $\vee_1, \wedge_1, \rightarrow_1, \leftrightarrow_1$, and $-_1$, that implement the logical connectives. We will use these functions in cases where it is needed (e.g. while talking about truth values of propositions). More generally, any propositional formula with n variables defines a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ (later, we will also see that any Boolean function can be expressed using a propositional formula).

We also define two special logical formulas. The formula $\top \equiv p \vee \neg p$, which is always true, and the formula $\bot \equiv p \wedge \neg p$ which is always false.

We can now define the truth assignment and the truth value of

formula more formally.

Definition 2. A *truth assignment* is a function $v : \mathbb{P} \to \{0, 1\}$, that is $v \in {}^{\mathbb{P}}2$.

A *truth value* $\bar{v}(\varphi)$ of a propositional formula φ for a truth assignment v is defined inductively as:

- $\bar{v}(p) = v(p)$ if $p \in \mathbb{P}$
- $\bar{v}(\neg\varphi) = -_1(\bar{v}(\varphi))$
- $\bar{v}(\varphi \vee \psi) = \vee_1(\bar{v}(\varphi), \bar{v}(\psi))$
- $\bar{v}(\varphi \wedge \psi) = \wedge_1(\bar{v}(\varphi), \bar{v}(\psi))$
- $\bar{v}(\varphi \to \psi) = \to_1 (\bar{v}(\varphi), \bar{v}(\psi))$
- $\bar{v}(\varphi \leftrightarrow \psi) = \leftrightarrow_1 (\bar{v}(\varphi), \bar{v}(\psi))$

We can easily show (by the induction on the structure of the formula) that the truth value of a formula φ depends only on the truth assignment of variables from $\mathrm{var}(\varphi)$.

A proposition φ over $\mathbb{P}$ is *true in (satisfied by) an assignment* $v \in {}^{\mathbb{P}}2$, if $\bar{v}(\varphi) = 1$. In such a case, v is called a *satisfying assignment* for φ, we denote this fact $v \vDash \varphi$. If the formula is true for all assignments $v \in {}^{\mathbb{P}}2$, we say that it is *valid (a tautology)* and denote the fact as $\vDash \varphi$. On the other hand, if there is no assignment for which the formula is true, it is called *unsatisfiable (a contradiction)*. A formula φ is *independent (a contingency)* if it is neither a tautology nor a contradiction, i.e. there are two assignments $v_1, v_2 \in {}^{\mathbb{P}}2$, such that $\bar{v}_1(\varphi) = 1$ and $\bar{v}_2(\varphi) = 0$. Finally, a formula is *satisfiable* if there is a truth assignment in which it is true.

A truth assignment of $\mathbb{P}$ is also called a model of the language $\mathbb{P}$. The set of all models of $\mathbb{P}$ is denoted as $M(\mathbb{P})$, and, obviously $M(\mathbb{P}) = {}^{\mathbb{P}}2$. A proposition φ over $\mathbb{P}$ is valid in a model $v \in M(\mathbb{P})$, if $\bar{v}(\varphi) = 1$. Then we also say that v is a model of φ, denoted as $v \vDash \varphi$. $M^{\mathbb{P}}(\varphi) = \{v \in M(\mathbb{P}) | v \vDash \varphi\}$ is the *class of all models* of φ. A formula is valid, if it is true in every model of the language, it is unsatisfiable if it does not have a model, and satisfiable if it has a model. It is independent if it is true in a model of the language and false in another one. Formulas φ and ψ are logically equivalent ($\varphi \sim \psi$), if $M^{\mathbb{P}}(\varphi) = M^{\mathbb{P}}(\psi)$.

The last two paragraphs say basically the same, the difference is that in the latter one, we use the notion of model, which is central to logic. The notion of models, and sets of models will be important later, and "model" is one of the key terms in logic.

In the definition of propositions, we used 5 different logical connectives. However, if we take a look at the table with their semantics, we may notice, that, for example, $p \to q$ is equivalent $\neg p \vee q$. Therefore, even without using the implication ($\to$) we can still express everything we could with them. More formally, for every formula $\varphi \in \mathrm{VF}_{\mathbb{P}}$, there is an equivalent formula φ' that does not use the implication. Moreover, we can notice, that $p \leftrightarrow q$ is equivalent to $(p \to q) \wedge (q \to p)$, therefore we even do not need the equivalence, and every formula can be written using only negation, conjunction, and disjunction ($\neg, \wedge, \vee$). This feature of the set can be defined more formally.

Definition 3. A set of connectives is *adequate* if they can express any Boolean function by some proposition from them.

We have already discussed that the set $\{\neg, \wedge, \vee\}$ is adequate. We can also show, that the set $\{\rightarrow, \neg\}$ is adequate, the easiest way to do that is to realize, that $(p \wedge q) \sim \neg(p \rightarrow \neg q)$ and $(p \vee q) \sim (\neg p \rightarrow q)$.

Generally, we can also define custom connectives, for example, the so called Shaffer stroke (NAND) is defined as $p \uparrow q \sim \neg(p \wedge q)$, or the Pierce arrow (NOR) is defined as $p \downarrow q \sim \neg(p \vee q)$. Interestingly, both $\{\uparrow\}$ and $\{\downarrow\}$ are adequate sets. This is an important fact for the construction of logical circuits as we can use a logical gate of only one kind (either NAND or NOR) to express any Boolean function.

Normal Forms

There are also special forms of formulas, which are often used. Among the most common ones are so called conjunctive and disjunctive normal forms. In order to define these two forms, we first need to define a literal. A *literal* is a propositional variable or its negation, for example, if $\mathbb{P} = \{p, q\}$ then all the literals we can construct over $\mathbb{P}$ are $\{p, \neg p, q, \neg q\}$. A formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals. Disjunctions of literals are also called *clauses*, therefore we can also say, that a CNF formula is a conjunction of clauses. On the other hand, a formula is in disjunctive normal form (DNF) if it is a disjunction of conjunctions of literals. So, for example, $(p \vee \neg q \vee r) \wedge (p \vee q) \wedge (\neg p \vee q \vee r)$ is a formula in CNF and$(\neg p \wedge q \wedge \neg r) \vee (\neg p \wedge \neg q) \vee (p \wedge \neg q \wedge \neg r)$ is a formula in DNF (and, moreover a negation of the previous one in CNF).

Now, we would like to show, that for every formula, there is an equivalent formula in CNF and another equivalent formula in DNF. To this end, we will need the following set of rules, which can be proven by checking the truth table of the propositional connectives:

1. $(\varphi \rightarrow \psi) \sim (\neg\varphi \vee \psi), (\varphi \leftrightarrow \psi) \sim ((\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi))$
2. $\neg\neg\varphi \sim \varphi, \neg(\varphi \wedge \psi) \sim (\neg\varphi \vee \neg\psi), \neg(\varphi \vee \psi) \sim (\neg\varphi \wedge \neg\psi)$
3. $(\varphi \vee (\psi \wedge \chi)) \sim ((\psi \wedge \chi) \vee \varphi) \sim ((\varphi \vee \psi) \wedge (\varphi \vee \chi))$
4. $(\varphi \wedge (\psi \vee \chi)) \sim ((\psi \vee \chi) \wedge \varphi) \sim ((\varphi \wedge \psi) \vee (\varphi \wedge \chi))$

We can also easily show (again by induction on the structure of the formula) that if we have a formula φ' which is obtained from φ by replacing some occurrences of its sub-formula ψ with an equivalent sub-formula ψ', then $\varphi \sim \varphi'$.

And finally, we can show the following theorem.

Theorem 2. *For every formula φ over $\mathbb{P}$, there are formulas φ_C and φ_D, such that φ_C is in CNF, φ_D is in DNF and $\varphi \sim \varphi_C$ and $\varphi \sim \varphi_D$.*

Proof. The propositions φ_C and φ_D can be obtained from φ by applying the rules 1 to 4 mentioned above. □

The discussion above shows one of the ways to obtain equivalent formulas in CNF and DNF to a given formula. We can in fact apply

the rules in the order, in which they are presented. First, we remove all the implications and equivalences by using the rules no. 1. Then, we move all negations to the literals (i.e. there are no negations outside of parentheses), using the rule no. 2 and, finally, we repeatedly apply rules no. 3 and 4 to obtain the CNF and DNF.

This syntactic approach is not the only one to obtain CNF/DNF from a given formula. We can also construct the truth table of the formula and then read the CNF/DNF almost directly from the table. We will show a more general approach here, we will construct a CNF and DNF formulas φ_C and φ_D such that $M^{\mathbb{P}}(\varphi_C) = M^{\mathbb{P}}(\varphi_D) = K \subseteq M(\mathbb{P})$, for a given finite set of truth assignments K over a finite $\mathbb{P}$.

Before we show the construction, we will define the notion of p^t for a variable p and a truth value t as

$$p^t = \begin{cases} p & \text{if } t = 1 \\ \neg p & \text{if } t = 0 \end{cases} .$$

Now, we can easily see that for a single assignment $v \in K$, the set of models of the formula $\bigwedge_{p \in \mathbb{P}} p^{v(p)}$ contains only v. For a set of assignments K, we can just make a disjunction over all assignments in K (remember K is a finite set). Therefore,

$$M(\bigvee_{v \in K} \bigwedge_{p \in \mathbb{P}} p^{v(p)}) = K .$$

Thus we constructed a formula in DNF whose models are exactly the set K.

Constructing a formula φ in CNF such that $M(\varphi) = K$ for some given finite K is slightly more complex. However, we can use the fact that the negation of a formula in DNF is a formula in CNF. Negating a formula in CNF/DNF means changing all the conjunctions to disjunctions and vice versa and changing all literals to the complementary ones (i.e. changing p to $\neg p$ and vice versa). So, we start by creating a formula $\neg\varphi$ in DNF for the set ${}^{\mathbb{P}}2 \setminus K$ according to the approach above. Then, we negate the formula, thus obtaining φ in CNF such that $M(\varphi) = K$. Following these two steps we obtain the CNF formula

$$\varphi = \bigwedge_{v \in {}^{\mathbb{P}}2 \setminus K} \bigvee p^{-1 v(p)}$$

such that $M(\varphi) = K$.

If we want to use this approach to create a formula in CNF or DNF equivalent to a formula φ, we simply choose $K = M(\varphi)$. This description also shows that any Boolean function f (i.e. function $f : \{0,1\}^n \to \{0,1\}$) can be expressed as a proposition. We can choose $K = \{v | f(v) = 1\}$.

Both the techniques described above lead to an equivalent formula in CNF/DNF, the table-based method is typically used only for formulas with lower number of variables, as the size of the table for a formula with n variables is 2^n.

Logical theories

In mathematics, we often need to work in theories – we assume that some facts are true (the axioms of the theory) and are interested in which other facts are true. Therefore, in logic, we define a *propositional theory over the language* $\mathbb{P}$ as a set of propositions from $\mathrm{VF}_{\mathbb{P}}$. These propositions are called *axioms*. An assignments $v \in M(\mathbb{P})$ is a *model of theory* T ($v \vDash T$), if all axioms of T are true in v. Similarly to formulas, we define the *class of models of* T as $M(T) = \{v \in M(\mathbb{P}) | v \vDash \varphi \text{for all} \varphi \in T\}$. A finite theory is equivalent to a conjunction of its axioms. We will also write $M(T, \varphi)$ as a shortcut for $M(T \cup \{\varphi\})$.

We can now re-define the semantics concepts with respect to a theory. Let T be a theory over $\mathbb{P}$ and φ a proposition over $\mathbb{P}$. We say that φ *is true (valid) in* T ($T \vDash \varphi$) it if is true in every model of T. In such a case, we also say that φ is a (semantic) consequence of T. A formula φ is *unsatisfiable (contradictory) in* T *(inconsistent with* T*)*, if it is false in every model of T. It is *independent (a contingency) in* T, if it is true is some model of T and false in another model of T and satisfiable in T, if it true in some model of T. Two propositions φ and ψ are *equivalent in* T *(*T*-equivalent)* ($\varphi \sim_T \psi$), if for every model $v \in M(T)$, $v \vDash \varphi$ if and only if $v \vDash \psi$. For an empty theory ($T = \emptyset$), or for a theory where all axioms are tautologies, the re-definitions in this paragraph are equivalent to the definitions mentioned earlier.

The concepts defined above can also be expressed using the sets of models. For example $T \vDash \varphi$ is the same as $M(T) \subseteq M(\varphi)$, and $\varphi \sim_T \psi$ is equivalent to $M(T, \varphi) = M(T, \psi)$.

For each theory, we define its consequence as the set of all propositions that are true in the theory – $\theta^{\mathbb{P}}(T) = \{\varphi \,| \varphi \in \mathrm{VF}_{\mathbb{P}}, T \vDash \varphi\}$. Now, if we have two theories T and T', such that $T \subseteq T'$ over $\mathbb{P}$, we can prove that $T \subseteq \theta^{\mathbb{P}}(T) = \theta^{\mathbb{P}}(\theta^{\mathbb{P}}(T)) \subseteq \theta^{\mathbb{P}}(T')$. The first part says, that each axiom of T is always a consequence of T. This makes sense, as an axiom of T is by definition true on all models of T. The next part says, that the consequences of consequences of T are still the original consequences. However, this is also simple to show. Obviously, $M^{\mathbb{P}}(T) = M^{\mathbb{P}}(\theta(T))$ and therefore also $\theta^{\mathbb{P}}(T) = \theta^{\mathbb{P}}(\theta^{\mathbb{P}}(T))$ by definition of the consequence. Finally, if we have a formula φ which is valid in all models of T then φ is also valid in all models of T' ($T \subseteq T'$) as each model of T' also must be a model of T. Therefore $\theta^{\mathbb{P}}(T) \subseteq \theta^{\mathbb{P}}(T')$.

Similarly, if we have propositions $\varphi, \varphi_1, \varphi_2, \ldots \varphi_n$ over $\mathbb{P}$, we can show that $\varphi \in \theta^{\mathbb{P}}(\{\varphi_1, \ldots, \varphi_n\})$ if and only if $\vDash (\varphi_1 \wedge \cdots \wedge \varphi_n) \rightarrow \varphi$.

A theory T over $\mathbb{P}$ is *inconsistent (unsatisfiable)*, if $T \vDash \bot$, otherwise T is *consistent (satisfiable)*. A theory is consistent if and only if it has a model. A theory is complete, if it is consistent and $T \vDash \varphi$ or $T \vDash \neg\varphi$ for every $\varphi \in \mathrm{VF}_{\mathbb{P}}$, i.e. there are no independent propositions in T. This is also equivalent to the fact that T has exactly one model (if T had two models v_1 and v_2, they there would be a propositional variable p, such that $v_1(p) \neq v_2(p)$ and therefore the formula p is true in one of the models and false in the other one, thus p is independent). In mathematics, we very often create new theories by adding axioms to

other theories. Such new theories are called extensions of the original theories. More formally, a theory T over $\mathbb{P}$ is an *extension* of T' over $\mathbb{P}'$, if $\mathbb{P}' \subseteq \mathbb{P}$ and $\theta^{\mathbb{P}'}(T') \subseteq \theta^{\mathbb{P}}(T)$, an extension is *simple*, if $\mathbb{P} = \mathbb{P}'$, and it is *conservative* if $\theta^{\mathbb{P}'}(T') = \theta^{\mathbb{P}}(T) \cap \mathrm{VF}_{\mathbb{P}'}$. Two theories T and T' are equivalent, if T is an extension of T' and vice versa.

Although we motivated the notion of extension by adding new axioms, it is defined more generally using the sets of consequences of the theory. This abstracts from the particular axioms and considers all equivalent theories the same. The notion of extension can also be expressed with the sets of models, if both theories T and T' are over the same language $\mathbb{P}$. In such a case T is an extension of T', if and only if $M^{\mathbb{P}}(T) \subseteq M^{\mathbb{P}}(T')$ and the two theories are equivalent if $M^{\mathbb{P}}(T) = M^{\mathbb{P}}(T')$.

We will conclude this section by the discussion about the number of nonequivalent propositions and theories over a finite language $\mathbb{P}$. We defined two formulas or theories equivalent, if they have the same sets of models. Therefore, if we want to compute the number of non-equivalent theories/formulas, we can instead compute the number of sets of models. So, if $|\mathbb{P}| = n$, than there are 2^{2^n} non-equivalent formulas (theories) over $\mathbb{P}$, as there are 2^n different assignments, and every set of assignments represents a formula (remember, we know how to write that formula in CNF/DNF) or a theory.

Using a similar reasoning, we can show how many nonequivalent valid (or contradictory – the number is the same) propositions are there in a theory. A valid proposition is true in all models of T, therefore there are $2^n - |M(T)|$ assignments where a valid proposition can be (but does not have to be) true. This means there are $2^{2^n - |M(T)|}$ valid (and contradictory) propositions in T. Every proposition is either valid, contradictory or independent, therefore there are $2^{2^n} - 2 \times 2^{2^n - |M(T)|}$ nonequivalent independent propositions in T. A theory has $2^{|M(T)|}$ simple extensions, one of these is contradictory (the set of models of an extension are a subset of the models of the original theory), and the same theory has $|M(T)|$ simple complete extensions (those correspond to single-element subsets of $M(T)$).

Instead of talking about nonequivalent propositions, we can also discuss T-nonequivalent propositions. There are $2^{|M(T)|}$ T-nonequivalent propositions (we know consider only subsets of $M(T)$ as the possible sets of models for the proposition), one of them is valid and one is contradictory in T, thus the number of T-nonequivalent propositions in T is $2^{|M(T)|} - 2$.

The fact, that we can use the number of subsets of models while computing the number of nonequivalent theories or formulas is more formally explained by so called Lindenbaum-Tarski algebra. For a consistent theory T over $\mathbb{P}$, we can define operations $\neg, \wedge, \vee, \bot, \top$ on the quotient set $\mathrm{VF}_{\mathbb{P}} \,/\, \sim_T$ by use of representatives, e.g. $[\varphi]_{\sim_T} \wedge [\psi]_{\sim_T} = [\varphi \wedge \psi]_{\sim_T}$. Then $AV^{\mathbb{P}}(T) = \langle \mathrm{VF}_{\mathbb{P}} \,/\, \sim_T, \neg, \wedge, \vee, \bot, \top \rangle$ is *Lindenbaum-Tarski algebra for* T. Since $\varphi \sim_T \psi$ if and only if $M(T, \varphi) = M(T, \psi)$ then $h([h]_{\sim_T}) = M(T, \varphi)$ is an injective function $h : \mathrm{VF}_{\mathbb{P}} \to \mathcal{P}(M(T))$. If $M(T)$ is finite then, h is additionally surjective, and

therefore AV is isomorphic to the algebra of sets $\mathcal{P}(M(T))$.

Satisfiability of Propositional Formulas

The problem of satisfiability of logical formulas is one of the central problems in computer science. The general question posed by the problem is, whether a given formula in CNF is satisfiable. In general, this problem is NP-complete[7], which means that we do not know any polynomial-time algorithm to solve it. However, there are some special types of formulas, for which SAT can be solved in polynomial time. In this section, we will discuss such formulas and show the algorithms that solve SAT for these, we will also briefly discuss local search algorithms for SAT, and describe the complete (but generally exponential) DPLL procedure.

[7] A problem c is NP-complete, if it is NP and if any other NP problem is reducible to c in polynomial time. A problem is NP if given a candidate solution, we can check in polynomial time that it is indeed a solution. A problem p is reducible to c in polynomial time, if we can transform each instance of c into an instance of p in polynomial time.

The first class of formulas are so called 2-CNF. A formula is in *k-CNF* if it is a disjunction of clauses and each of the clauses contain at most k literals. The SAT problem for k-CNF formulas is called the k-SAT. The k-SAT problem is NP-complete for $k > 2$, however for $k = 2$ it can be solved in polynomial time using the so called implication graph of the formula.

Definition 4. Let φ is a formula over $\mathbb{P}$ in 2-CNF, $\varphi \equiv \bigwedge_{i=1}^{j}(l_{i1} \vee l_{i2}) \wedge \bigwedge_{i=j+1}^{k} l_i$. The *implication graph* of φ is a oriented graph $G_\varphi = (V, E)$, where the set of vertices is

$$V = \{p | p \in \mathbb{P}\} \cup \{\neg p | p \in \mathbb{P}\}$$

and the set of edges is

$$E = \{(\bar{l}_{i1}, l_{i2}) | 1 \leq i \leq j\} \cup \{(\bar{l}_{i2}, l_{i1}) | 1 \leq i \leq j\} \cup \{(\bar{l}_i, l_i) | j+1 \leq i \leq k\}\,.$$

In the implication graph, the set of vertices corresponds to all literals from variables in $\mathrm{var}(\varphi)$ and each clause in the formula is represented as one or two edges. For a clause $l_1 \vee l_2$, we include two edges (implications) $\bar{l}_1 \to l_2$ and $\bar{l}_2 \to l_1$. These two implications are logically equivalent to the clause. For a *unit clause* l (a clause with only a single literal), we include a single edge $\bar{l} \to l$, this is also equivalent to l. The implication graph thus contains the 2-CNF formula written as implications between its literals. The implication graph of a formula can be constructed in a time linear in the length of the formula.

Let us now assume that a truth assignment $v \in {}^{\mathbb{P}}2$ satisfies a formula φ. In such a case, in every strongly connected component[8] in G_φ, all the literals have the same truth value. Otherwise there would be an implication which is not true in the assignment which is a contradiction with the fact that the whole formula is true in the assignment[9]. This also means that if we have a satisfying assignment for φ, none of the strongly connected components contain both a literal and its negation.

[8] In a strongly connected component, there is an oriented path between every pair of vertices.

[9] Assume that literals l_1 and l_n are in the same strongly connected component and that $v(l_1) = 1$ and $v(l_n) = 0$. There is a chain of implications $l_1 \to l_2, l_2 \to l_3, \ldots, l_{n-1} \to l_n$, at least one of these must be $1 \to 0$ and therefore false.

Can we use the implication graph of a formula to obtain a satisfying truth assignment? We indeed can, but only if none of the strongly connected components contain a pair of complementary literals. In

such a case, we can contract each of the strongly connected components into a single vertex (thus obtaining a graph G^*_φ). Such a graph would be acyclic and therefore has a topological ordering $<$. We create an assignment v in a few steps: for every unassigned component in increasing order of $<$, we assign 0 to all its literals and 1 to all the complementary literals in the graph (they would in fact also form an strongly connected component). Such an assignment is satisfying for φ. If not, G^*_φ would contain edges $p \to q$ and $\bar{p} \to \bar{q}$ with $v(p) = 1$ and $v(q) = 0$, but that contradicts the order of assignments as $p < q$ and $\bar{q} < \bar{p}$.

The discussion above can be summarized in the theorem bellow.

Theorem 3. *Proposition φ in 2-CNF is satisfiable if and only if no strongly connected component of its implication graph G_φ contains a pair of complementary literals.*

As the implication graph can be constructed in linear time and the strongly connected components can also be found in linear time. This also shows that the 2-SAT problem can be solved in linear time.

Another class of formulas where SAT can be solved in polynomial time are conjunctions of clauses with at most one positive literal. Such clauses are called Horn clauses, and such formulas are called Horn formulas. The problem of satisfiability of Horn formulas is called Horn-SAT.

The Horn clauses can also be interpreted as implications. The Horn clause $(\neg p_1 \vee \neg p_2 \vee \cdots \vee \neg p_n \vee q)$ is equivalent to the implication $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \to q$.

Deciding whether a Horn formula φ is satisfiable or not is simple, and can be done using the following algorithm:

1. If φ contains a pair of unit clauses l and $\bar{l}$ (a pair of complementary literals) it is not satisfiable.

2. If φ contains a unit clause l, assign 1 to l, remove all clauses containing l, remove $\bar{l}$ from all clauses, and continue from the start.

3. If φ does not contain a unit clause, it is satisfied by assigning 0 to all remaining propositional variables.

The first step of the algorithm is obviously correct, as $p \wedge \neg p$ is a contradiction, and the last step follows from the form of Horn formulas, as each of the remaining clauses contains at least one negative literal. It remains to show that the second step (also called *unit propagation*) is also correct. The formula φ can be satisfied only if each of its clauses is true, and therefore the unit clause l must also be true. Once we assign 1 to l, we can remove all the clauses that contain l (these are already satisfied) and we can also remove $\bar{l}$ from all the remaining clauses as $\bar{l}$ is 0, and therefore the clauses need to be satisfied by other literals.

This shows, that Horn-SAT can be solved in polynomial time. The direct implementation of the algorithm described above is quadratic, however there are even linear implementations.

We already mentioned that there are no polynomial algorithms for the SAT problem in general, but we can use some local search algorithms to attempt to solve the problem. For example, the GSAT algorithm starts with a random truth assignment. If this assignment satisfies the formula, the algorithm ends. Otherwise it flips the truth value for one of the variables – it chooses the variable whose change leads to the smallest number of unsatisfied clauses in the new assignment. There is a small chance to change a random variable (this allows the algorithm to escape local minima in the number of unsatisfied clauses). The WalkSAT algorithm works in a similar way, but instead of picking a variable from the whole formula, it first selects a random clause and picks a variable from it which minimizes the number of previously satisfied clauses that become unsatisfied by the change. It also has a small chance to pick a variable at random.

While none of these algorithms can guarantee that they find the satisfying assignment if it exists, they are very fast, and very often can indeed find the satisfying assignment.

A complete algorithm (such that always finds the assignment, if it exists) can be implemented using backtracking and testing all possible assignments, however, such an algorithm is generally exponential.

The DPLL procedure implements such a backtracking with some improvements. It first removes all clauses that are tautologies, than if a clause becomes empty during the run of the algorithm, it indicates that the current partial assignment cannot satisfy the formula and the DPLL procedure fails. After these simple steps, the DPPL procedure simplifies the formula using unit propagation and so called *pure literal elimination*[10]. If none of the previous step can be applied, the DPLL procedure uses a splitting rule – it selects a literal and tries to call the DPLL procedure twice. Once for each possible truth assignment of that literal. If at least one of these calls succeeds, the formula is satisfiable.

[10] if a literal l is only positive or only negative in the formula, it can be assigned such value $v(l) = 1$ and all the clauses containing it can be removed

Contents

List of Figures

List of Tables

List of Algorithms

Todo list