Introduction to Mathematical Logic, Edition 2021

Book · February 2021

CITATIONS
0

READ
1

2 authors, including:

Karlis Podnieks
University of Latvia
75 PUBLICATIONS **273** CITATIONS

SEE PROFILE

Version released: February 7, 2021

This edition is dedicated to the memory of
Professor **Elliott Mendelson**, 1931-2020

Introduction to Mathematical Logic

Textbook for students

Edition 2021

by Vilnis Detlovs, Dr. math.,
and Karlis Podnieks, Dr. math.

University of Latvia

Sections 1, 2, 3 of this book represent an extended translation of the corresponding chapters of the book: **V. Detlovs, Elements of Mathematical Logic,** Riga, University of Latvia, 1964, 252 pp. (in Latvian). With kind permission of Dr. Detlovs.

Vilnis Detlovs, 1923-2007. Memorial Page

This textbook contains links to:
Wikipedia, the free encyclopedia;
MacTutor History of Mathematics archive of the University of St Andrews;
MathWorld of Wolfram Research.

Table of Contents

References

Detlovs V. [1964] Elements of Mathematical Logic, *University of Latvia*, 1964, 252 pp. (in Latvian).

Hilbert D., Bernays P. [1934] Grundlagen der Mathematik. Vol. I, Berlin, 1934, 471 pp. (Russian translation available)

Kleene S.C. [1952] Introduction to Metamathematics. *Van Nostrand*, 1952 (Russian translation available)

Kleene S.C. [1967] Mathematical Logic. *John Wiley & Sons*, 1967 (Russian translation available)

Mendelson E. [1997] Introduction to Mathematical Logic. Fourth Edition. *International Thomson Publishing*, 1997, 440 pp. (Russian translation available)

Podnieks K. [1997] What is Mathematics: Gödel's Theorem and Around. 1997-2015 (see *ResearchGate* for English and Russian online versions).

1. Introduction. What Is Logic, Really?

Attention! In this book,
predicate language is used as a synonym of **first order language**,
formal theory – as a synonym of **formal system, deductive system,**
predicate logic – as a synonym of **first order logic without equality.**
constructive logic – as a synonym of **intuitionistic logic,**
algorithmically solvable – as a synonym of **recursively solvable,**
algorithmically enumerable – as a synonym of **recursively enumerable.**

1.1. Total Formalization is Possible!

What is logic? Of course, logic is "about reasoning". Parts of our knowledge may be inter-dependent, so, one part may be derived from some other ones. A trivial example:

All fathers are male persons.

No person can be male and female simultaneously.

Miranda is a female person.

Hence, Miranda is not a father.

Of course, we knew the latter "fact" in advance. But imagine, we are trying to teach this kind of reasoning to a computer. A computer may know in advance only that part of our knowledge that is stored in its **knowledge base**. But, it is impossible to store *all* of our knowledge. Thus, we must avoid storing of the knowledge that is easily derivable from the one already stored. For example, if the first three of the above propositions have been already stored, then we need not to store the fourth one – it can be derived by reasoning:

Assume, Miranda is a father. Then Miranda is a male person. But Miranda is a female person. Hence, Miranda is male and female simultaneously. This is impossible. Hence, Miranda is not a father.

Therefore, we must implement on our computer not only the knowledge base, but also the necessary means of reasoning (an advanced kind of **query processing**).

Hence, today, we have a very fundamental reason to **formulate** our knowledge

and means of reasoning **explicitly**: only in this way we can transmit the knowledge and the ability of reasoning to computers.

Of course, explicit reasoning started long before computers – probably, in the 6th century BC when Greeks proved the first mathematical theorems. Let us consider one of them:

Theorem. There are more prime numbers than any prescribed amount. (In modern terms: there are infinitely many prime numbers.)

Proof (modern notation is used). Assume the contrary, that $p_1, \ldots, p_k$ is the complete list of all primes, and consider the number $N = p_1 \cdot \ldots \cdot p_k + 1$. We know that such N is divisible by none of $p_1, \ldots, p_k$. But we know also that any (natural) number is divisible by a prime. So, N must be divisible by a prime that does not belong to the alleged "complete" list. Q.E.D.

Here, the statement of Theorem is derived from other statements that "we know". How do we know them? Either we or other people have proved these statements earlier – by deriving them from some other statements, or, they were forced to adopt them without proof, for example, as "obvious" ones. Indeed, asking for proofs over and again indefinitely long time is hopeless. At some moment, such a process must be stopped – by a decision to adopt some of the statements without proof, as "axioms".

Additional stimulus to explicit reasoning was a contradiction found in geometrical reasoning. The early Pythagoreans arrived at an implicit belief that any two line segments must possess a common measure. Namely (in modern terms), if x and y are lengths of two line segments, then there exists a (usually, smaller) segment z such that $x = pz\,;\, y = qz$, where p, q are some natural numbers. Of course, if z is the *maximum* common measure, then p, q possess no common divisors. Now, let us try to find the common measure of x and y being the side and the diagonal of a square. By Pythagorean Theorem: $x^2 + x^2 = 2x^2 = y^2$. But there is a maximum z such that

$$x = pz\,;\, y = qz\,;\, 2p^2 z^2 = q^2 z^2\,;\, 2p^2 = q^2 \quad .$$

Hence, q is even: $q = 2r\,;\, q^2 = 4r^2\,;\, p^2 = 2r^2$. Thus, p is even as well, but we know that p and q do not possess common divisors. We have arrived at a contradiction! According to a legend, this caused a serious conflict among Pythagoreans. The solution was found by abandoning the belief in the universal existence of common measures, and by re-building the geometry accordingly.

Note. Of course, in modern terms, Pythagoreans discovered that $\sqrt{2}$ is an irrational number, i.e., that there are no natural numbers p, q such that $\sqrt{2} = \frac{q}{p}$ (or, $2p^2 = q^2$, see above).

How far can people proceed with explication? Can *any* implicit knowledge (intuition) be converted into an explicit knowledge represented as a list of axioms? Can *any* sophisticated human skills of reasoning be converted into an explicit list of rules of inference? Is this really possible?

It took more than 2000 years to develop the necessary methods. The process ended in 1870s and subsequent decades – in the work of Georg Cantor, Gottlob Frege, Charles S. Peirce, Bertrand Russell, David Hilbert, of their colleagues and followers. Today, the logical techniques developed by these people allow for an axiomatic reconstruction of any theory that is based on a stable system of principles (in particular, of any mathematical theory).

Total axiomatic reconstruction is called **formalization**. The results of such reconstruction are called **formal theories** (the terms “formal systems” and “deductive systems” also are used) emphasizing that in these theories no step of reasoning can be done without a reference to an exactly formulated list of axioms and rules of inference. Even the most "self-evident" logical principles (like as, "if *A* implies *B*, and *B* implies *C*, then *A* implies *C*") must be either formulated in the list of axioms and rules explicitly, or must be derived from it.

Modern students can think of formal theories as **knowledge bases**. In these bases, knowledge is stored in the form of propositions, also called **axioms**. The simplest kind of axioms are **facts** representing data of the usual databases, for example, personal data of people, such as “*Miranda is a female person*”. The second kind of axioms are **rules** representing data constraints, such as “*All fathers are male persons*”. Knowledge bases are equipped with **query processing software.** However, the situation with query processing is here more complicated than in the case of the usual databases. For example, imagine, we would wish to ask “Is Miranda a father?” This would mean, in fact, asking the question: which of the propositions, “*Miranda is a father*” or “*Miranda is not a father*” follows from the axioms stored in the knowledge base?

The first distinctive feature of a formal theory (or, knowledge base) must be a **precisely defined** ("formal") **language** used to express its **propositions**. "Precisely defined" means here that there is an algorithm allowing to determine, **is a given character string a correct proposition, or not**. (Algorithms are mechanically applicable procedures that do not refer to implicit knowledge or human skills. Algorithms can be implemented as computer programs. For more details, see Algorithm in Wikipedia)

The second distinctive feature of a formal theory must be a **precisely defined** ("formal") **notion of proof**. Each proof proves some proposition, that is called (after being proved) a **theorem**. Thus, theorems form a subset of propositions.

What could mean here “precisely defined”? Curiously enough, the most

general definition is very simple, and it does not mention neither axioms, nor rules of inference: precisely defined notion of proof means that there is an algorithm allowing to determine, **is a given text a correct proof, or not**.

Neither "self-evident" axioms, nor "plausible" rules of inference are distinctive features of the "formality". Speaking strictly, "self-evident" is synonymous to "accepted without argument". Hence, axioms and/or rules of inference may be "good, or bad", "true, or false", and so may be the theorems obtained by means of them. The only definitely verifiable thing is here the fact that some theorem has been, indeed, **proved** by using some definite set of axioms, and by means of some definite set of inference rules. And this fact must be verifiable by means of an algorithm – by a procedure that can be implemented on a computer.

Thus, a theory *T* is called a formal theory if and only if there are two algorithms:

a) an algorithm allowing to verify, is a given character string a correct proposition of T, or not;

b) an algorithm allowing to verify, is a given text a correct proof according to the principles of *T*, or not.

If somebody is going to publish a "mathematical text" calling it "*proof of a theorem in theory T*", then we must be able to verify whether the text in question is really a correct proof according to the standards of proving accepted in theory *T*. Thus, in a formal theory, the standards of reasoning should be defined precisely enough to implement verification of proofs on a computer. But note that we are discussing here **verification of ready proofs**, and not the much more difficult problem – **is some proposition provable** in *T* or not, see below.

Axioms and rules of inference represent the most popular of the possible techniques of formalization (see Exercise 1.1.7 below).

Chess game as a formal “theory”

As an unpractical example of a formal theory let us consider the *game of chess*, let us call this "theory" *CHESS*. Let 's define as **propositions** of *CHESS* all the possible positions – i.e., allocations of some of the pieces (kings included) on a chessboard – plus the flag: "white to move" or "black to move". Thus, the set of all the possible positions represents the **language** of *CHESS*. The only **axiom** of *CHESS* is the initial position (“white to move” included), and the **rules of inference** – the well-known *rules of the game*. Rules define “proof steps” – they allow correct passing from some propositions of *CHESS* to some other ones. Starting with the axiom and iterating moves allowed by the rules we obtain **theorems** of *CHESS*. Thus, theorems of *CHESS* are defined as all the possible positions (i.e., propositions of *CHESS*) that can be

obtained from the initial position (the axiom of *CHESS*) by moving pieces according to the rules of the game (i.e., by using the inference rules of *CHESS*). Note that here, correct silly moves are considered as correct “proof steps”.

Exercise 1.1.1 (optional). Could you provide an **unprovable** proposition of *CHESS*? Try proving that your proposition is unprovable, indeed, i.e., that your position cannot be obtained from the initial position by moving pieces according to the rules of the game.

Note. By the way, as you see it now, and will see later again: in logic, “negative” proving, i.e., proving that something cannot be proved, may be a more complicated task than “positive” proving.

Why could *CHESS* be called a formal theory? When somebody offers a "mathematical text" *P* as a proof of a theorem *A* in *CHESS*, this means that *P* is a record of some chess-game stopped in the position *A*. Checking the correctness of such "proofs" is a boring, but an easy task. The rules of the game are formulated precisely enough – we could write a computer program that will execute the task.

Exercise 1.1.2 (optional). Try estimating the size of this program in some programming language.

Another toy example

Our second example of a formal theory is only a bit more serious. It was proposed by Paul Lorenzen, so let us call this theory *Lo*. **Propositions** of *Lo* are all the possible (non-empty) "words" made of letters *a, b,* for example: *a, b, aa, aba, baab*. Thus, the set of all these "words" represents the **language** of *Lo*. The only **axiom** of *Lo* is the word *a*, and *Lo* has two **rules of inference**:

$$X \vdash Xb;\ X \vdash aXa.$$

This means that (in *Lo*) from a proposition *X* we can infer immediately the propositions *Xb* and *aXa*. For example, the proposition *aababb* is a **theorem** of *Lo*:

$$a \underset{\text{rule1}}{\vdash} ab \underset{\text{rule2}}{\vdash} aaba \underset{\text{rule1}}{\vdash} aabab \underset{\text{rule1}}{\vdash} aababb$$

This fact is expressed usually as $Lo \vdash aababb$ ("Lo proves *aababb*", $\vdash$ being a "fallen T").

Exercise 1.1.3. a) Verify that *Lo* is a formal theory. (Hint: describe an algorithm allowing to determine, is a sequence of propositions of *Lo* a correct proof, or not.)

b) (P. Lorenzen) Verify the following property of theorems of Lo: for any *X*, if $Lo \vdash X$, then $Lo \vdash aaX$.

General properties of formal theories

One of the most important properties of formal theories is given in the following

Exercise 1.1.4. Show that the **set of all theorems of a formal theory is algorithmically enumerable**, i.e., show that, for any formal theory T, one can build an algorithm A_T that generates and prints out on an endless paper tape all theorems of this theory (and nothing else). (Hint: we will call *T* a formal theory if and only if we can present an algorithm for checking texts as correct proofs via principles of reasoning of *T*. Thus, assume, you have 4 functions: *GenerateFirstText()* – returns *Text*, *GenerateNextText()* – returns *Text*, *IsCorrectProof(Text)* – returns *true* or *false*, *ExtractTheorem(Text)* – returns *Text*, and you must implement the functions *GenerateFirstTheorem()* – returns *Text*, *GenerateNextTheorem()* – returns *Text*).

Unfortunately, **such generating algorithms cannot solve the problem that the mathematicians are mainly interested in: is a given proposition *A* provable in *T* or not?** Indeed, when, executing the algorithm A_T, we see our proposition *A* printed out, this means that *A* is provable in *T*. Still, in general, until that moment, we cannot know in advance whether *A* will be printed out some time later or it will not be printed at all.

Note. According to the official terminology, algorithmically enumerable sets are called "recursively enumerable sets", in some texts – also "listable sets".

Exercise 1.1.5. a) Describe an algorithm determining whether a proposition of *Lo* is a theorem or not.

b) (optional) Could you imagine such an algorithm for *CHESS*? Of course, you could, yet... Thus you see that even, having a relatively simple algorithm for *checking the correctness* of proofs, the problem of determining *provability* can be a very complicated one.

T is called a **solvable theory** (more precisely – **algorithmically solvable theory**) if and only if there is an algorithm allowing to check whether some proposition is *provable* by using the principles of *T* or not.

In the Exercise 1.1.5(a) you proved that *Lo* is a solvable theory. Still, in the Exercise 1.1.5(b) you established that it is hard to state whether *CHESS* is a "feasibly solvable" theory or not. **Determining the provability of propositions is a much more complicated task than checking the correctness of ready proofs.** It can be proved that **most mathematical theories are unsolvable**, the elementary (first order) arithmetic of natural numbers included (see, for example, Mendelson [1997], or Podnieks [1997], Section 6.3). There is no algorithm allowing to determine, is some arithmetical proposition provable from the axioms of arithmetic, or not.

Note. According to the official terminology, algorithmically solvable sets are called "recursive sets".

Normally, formal theories contain the negation symbol ***not***. In such theories, solving of the problem stated in a proposition *A* means proving either *A,* or proving ***not****A* ("disproving *A*", "refuting *A*"). We could try to solve the problem by using the enumeration algorithm of the Exercise 1.1.4: let us wait until *A* or ***not****A* is printed. In general, we have four possibilities here:

a) *A* will be printed, but ***not****A* will not (then the problem *A* has a positive solution);

b) ***not****A* will be printed, but *A* will not (then the problem *A* has a negative solution);

c) *A* and ***not****A* will be printed both;

d) neither *A*, nor ***not****A* will be printed.

In the case c), by using principles of *T,* some proposition and its negation can be proved simultaneously, i.e., T is an **inconsistent theory**. This means that the principles of T must be re-examined (to determine the cause of inconsistency) and corrected.

In the case d) we may be waiting forever, yet nothing interesting will happen: by using the principles of *T* one can neither prove nor disprove the proposition *A,* and for this reason such a theory is called an **incomplete theory**. The famous Incompleteness Theorem proved by Kurt Gödel in 1930 says that **most mathematical theories are either inconsistent or incomplete** (see Mendelson [1997] or Podnieks [1997], Section 6.1).

Exercise 1.1.6. Show that any (simultaneously) consistent and complete formal theory is solvable. (Hint: use the algorithm of the Exercise 1.1.4, i.e., assume that you have the functions *GenerateFirstTheorem()* − returns *Text*, *GenerateNextTheorem()* − returns *Text*, and implement the function *IsProvable(Text)* that returns *true* or *false*. Where the consistency and completeness come in here?)

Exercise 1.1.7 (optional). a) Verify that "fully axiomatic theories" are formal theories in the sense of the above general definition. (Hint: assume, that you have the following functions: *GenerateFirstText()* − returns *Text*, *GenerateNextText()* − returns *Text*, *IsPropositon(Text)* − returns *true* or *false*, *IsAxiom(Proposition)* − returns *true* or *false*, there is a finite list of inference rule names: $\{R_1, ..., R_n\}$, function *Apply(RuleName, ListOfPropositions)* − returns *Proposition* or *false*, and you must implement the functions *IsCorrectProof(ListOf Propositions)* − returns *true* or *false*, *ExtractTheorem(Proof)* − returns *Proposition*).

b) (for smart students) What, if, instead of $\{R_1, ..., R_n\}$, we would have an *infinite* list of inference rules, i.e., functions *GenerateFirstRule()*, *GenerateNextRule()* returning *RuleName*?

1.2. Predicate Languages

History

Aristotle (384-322 BC) – in a sense, the "first logician", "... was not primarily a mathematician but made important contributions by systematizing deductive logic." (according to MacTutor History of Mathematics archive).

Gottlob Frege: "In 1879 Frege published his first major work *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.* ... In this work Frege presented for the first time what we would recognise today as a logical system with negation, implication, universal quantification, essentially the idea of truth tables etc." (according to MacTutor History of Mathematics archive).

Hilary Putnam. Peirce the Logician. *Historia Mathematica*, Vol. 9, 1982, pp. 290-301 (an online excerpt available, published by John F. Sowa).

Imagine, we are trying to build a **knowledge base** (or, a **formal theory**) by making explicit some piece of our (until now – mainly implicit) knowledge. How should we proceed?

We have an informal vision of some domain consisting of "objects". When speaking about it, we are uttering various *propositions*, and some of these propositions we regard as "true" statements about the domain. Thus, our first formalization task must be defining of some **formal language**, allowing to put all our propositions about the domain in a uniform and precise way.

After this, we can start considering propositions that we are regarding as "true" statements about the domain. There may be an infinity of such statements, hence, we can't store into knowledge base all of them, so we must organize them somehow. Some minimum of the statements we will declare as **axioms**, and store in the knowledge base. The other ones we will try to derive from the axioms by using some precisely defined **rules of inference**.

Formulation of axioms (in knowledge bases they are called *facts* and *rules*) and rules of inference (i.e., logic) is an absolutely necessary next step, if we wish to transmit our knowledge to computers. In advance, computers "do not know" even the most trivial things about the human society, such as "persons cannot be parents of themselves". Or, about natural numbers: we must submit to our computer, for example, $x+0=x$ as an axiom, if we wish it to learn proving of mathematical theorems.

In mathematics and computer science, the most common approach to the first phase of formalization is using of the so-called **predicate languages**, first introduced by G. Frege and C. S. Peirce.

(In most textbooks, they are called **first order languages**, however, see below the warning about second order languages.)

Usually, we are taught to analyze the sentence "John loves Britney" as follows: *John* – subject, *loves* – predicate, *Britney* – object. The approach of predicate languages is different: instead of "John loves Britney", let us write

$$loves(John, Britney),$$

where *loves(x, y)* is a two-argument **predicate**, and *John*, and *Britney* are objects. By the way, following this principle literally, we should write *=(x, y)* instead of $x=y$.

Another example: in a predicate language, to say “All people are mortal”, we will write: “for all x, if x is a person, then x is mortal”, or,

$$\forall x\,(person(x) \rightarrow mortal(x)) \quad .$$

This approach – reducing of human language sentences to **variables, constants, functions, predicates and quantifiers**, appears to be extremely flexible, and it is much more uniform when compared to the variety of constructs used in natural human languages. A unified approach is much easier to use for communication with computers.

Yet another example: "Britney works for BMI as a programmer". In a predicate language, we must introduce a 3-argument predicate "x works for y as z", or *works(x, y, z).* Then, we may put the above fact as: *works(Britney, BMI, Programmer)*.

Note. Representing data in the form of predicates was the main idea of Edgar F. Codd when he introduced relational databases (“SQL databases”) in 1970. Database tables represent predicates.

Language primitives

The informal vision behind the notion of predicate languages is centered on the so-called "domain" – a non-empty collection of "objects", their "properties" and the "relations" between them, that we wish to describe by using the language. This vision serves as a guide in defining the formal language, and further – when selecting axioms and rules of inference.

Object variables

The first kind of language elements we will need are **object variables** (sometimes called also individual variables, or simply, variables). We need an unlimited number of them):

$$x, y, z, x_1, y_1, z_1,...$$

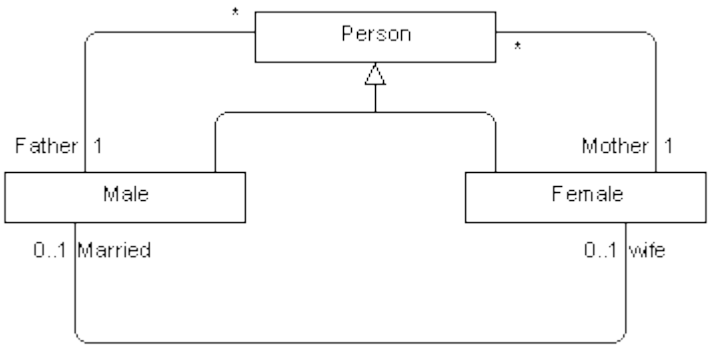
The above-mentioned "domain" is the intended “range” of all these variables.

Examples. 1) Building a language that should describe the "domain of people", we must start by introducing "variables for people": x denotes an arbitrary person.

2) Building the language of the so-called **first order arithmetic**, we are thinking about "all

natural numbers" as the range of variables: 0, 1, 2, 3, 4, …: *x* denotes here an arbitrary natural number.

3) Building the language of **set theory**, we think about "all sets" as the range of variables: *x* denotes an arbitrary set.



"*Domain of people" represented as a UML class diagram*

Note. Since our screens and printers allow only a limited number of pixels per inch, in principle, we should generate variable names by using a finite set of characters. This can be achieved, for example, by using a single letter x:

x, xx, xxx, xxxx, xxxxx,...

Object constants

The next possibility we may wish to have in our language are the so-called **object constants** (sometimes called individual constants, constant letters, or simply, constants) – names or symbols denoting some specific "objects" of our "domain".

Examples. 1) In our "language about people" we may introduce constants identifying particular people: John, Britney etc.

2) In the language of first order arithmetic, we may wish to introduce two constants – 0 and 1 to denote "zero" and "one" – two natural numbers having specific properties.

3) In the language of set theory, we could introduce a constant denoting the empty set, but there is a way to do without it as well (for details, Podnieks [1997], Section 2.3).

Function constants

In some languages we may need also the so-called **function constants** (sometimes called function letters) – names or symbols denoting specific functions, i.e., mappings between "objects" of our "domain", or operations on

these objects.

Examples. 1) In our "language about people" we will not use function constants.

2) In the language of first order arithmetic, we introduce two function constants "+" and "*" denoting the usual addition and multiplication of natural numbers, i.e., the two-argument functions x+y and x*y.

3) In the language of set theory, we could introduce function constants denoting set intersections $x \cap y$, unions $x \cup y$, set differences $x - y$, power sets $P(x)$ etc., but there is a way to do without these symbols as well (for details, Podnieks [1997], Section 2.3).

In mathematics, normally, we are writing *f(x, y)* to denote the value of the function *f* for the argument values *x, y*. This (the so-called "prefix" notation) is a uniform way suitable for functions having any number of arguments: *f(x), g(x, y), h(x, y, z)* etc. In our everyday mathematical practice some of the two-argument functions (in fact, operations) are represented by using the more convenient "infix" notation (*x+y, x*y* instead of the uniform *+(x, y), *(x, y)*, etc.).

Note. In a sense, object constants can be viewed as a special case of function constants – an object constant is a “zero-argument function”.

Predicate constants

The last (but the most important!) kind of primitives we need in our language are the so-called **predicate constants** (sometimes called predicate letters) – names or symbols denoting specific **properties** (of) or **relations** between "objects" of our "domain".

Note. Using "predicate" as the unifying term for "property" and "relation" may seem somewhat unusual. But some kind of such unifying term is necessary. Properties are, in fact, unary (i.e., one-argument) "predicates", for example, "*x* is red". Relations are, two- or more-argument "predicates", for example, "*x* is better than *y*", or "*x* sends *y* to *z*".

Examples. 1) In our "language about people" we will use the following predicate constants (see the class diagram above):

Male(x) − means "x is a male person";

Female(x) − means "x is a female person";

Mother(x, y) − means "x is mother of y";

Father(x, y) − means "x is father of y";

Married(x, y) − means "x and y are married";

x=y − means "x and y are the same person".

The first two constants represent, in fact, "properties" (or, "classes") of our objects. The other 4 constants represent "relations" between our objects. The term "predicate" is used to include

both versions. We do not introduce *Person(x)* as a separate predicate because our domains consists of persons only.

2) It may seem strange to non-mathematicians, yet the most popular relation of objects used in most mathematical theories, is **equality** (or identity). Still, this is not strange for mathematicians. We can select an object x in our "domain" by using a very specific combination of properties and relations of it, and then – select another object y – by using a different combination. And after this (sometimes it may take many years to do) we prove that x=y, i.e., that these two different combinations of properties and relations are possessed by a single object. Many of the discoveries in mathematics could be reduced to this form.

In the language of first order arithmetic, equality "=" is the only necessary predicate constant. Other "basic" relations must be reduced to equality. For example, the relation x<y for natural numbers x, y can be reduced to equality by using the addition function and the formula ∃z(x+z+1=y).

3) In the language of set theory a specific predicate constant "in" denotes the set membership relation: $x \in y$ means "x is a member of y". The equality predicate x=y also is used – it means "the sets x and y possess the same members".

The uniform way of representation suitable for predicates having any number of arguments is again the "prefix" notation: *p(x), q(x, y), r(x, y, z)* etc. In the real mathematical practice, some of the two-argument predicates are represented by using the "infix" notation (for example, x=y instead of the uniform =(x, y), etc.).

Zero-argument predicate constants? In an interpretation (see Section 4.1 below), each such predicate must become either "true", or "false". Hence, paradoxically, zero-argument predicate constants would behave like as "propositional variables" – they represent assertions that do not possess a meaning, but possess a "truth value".

Summary of language primitives

Thus, the specification of a predicate language includes the following **primitives**:

1) A countable set of object variable names.

2) An empty, finite, or countable set of object constants.

3) An empty, finite, or countable set of function constants. To each function constant a fixed argument number must be assigned.

4) A finite, or countable set of predicate constants. To each predicate constant a fixed argument number must be assigned.

Different sets of primitives yield different predicate languages.

Examples. 1) Our "language about people" is based on: a) object variables x, y, z, ...; b) object constants: John, Britney, ...; c) function constants: none; d) predicate constants: *Male(x), Female(x), Mother(x, y), Father(x, y), Married(x, y), x=y*.

2) The language of first order arithmetic is based on: a) object variables x, y, z, ...; b) object constants: 0, 1; c) function constants: x+y, x*y; d) predicate constant: x=y.

3) The language of set theory is based on: a) object variables x, y, z, ...; b) object constants: none; c) function constants: none; d) predicate constants: $x \in y$, $x = y$.

The remaining part of the language definition is common for all predicate languages.

Terms and formulas

By using the language primitives, we can build terms, atomic formulas and (compound) formulas.

Terms are expressions used to denote objects and functions:

a) Object variables and object constants (if any), are terms.

b) If f is a k-argument function constant, and t_1, ..., t_k are terms, then the string $f(t_1, ..., t_k)$ is a term.

c) There are no other terms.

Examples. 1) In our "language about people" only variables x, y, z, ..., and object constants *John, Britney, ...* are terms.

2) In the language of first order arithmetic, for addition and multiplication the "infix" notation is used: if t_1, t_2 are terms, then (t_1+t_2) and (t_1*t_2) are terms. Of course, the object constants 0, 1 and variables x, y, z, ... are terms. Examples of more complicated terms: (x+y), ((1+1)*(1+1)), (((1+1)*x)+1).

3) In the language of set theory, variables x, y, z, ... represent the only kind of terms.

If a term does not contain variable names, then it denotes an "object" of our "domain" (for example, ((1+1)+1) denotes a specific natural number – the number 3). If a term contains variables, then it denotes a function. For example, (((x*x)+(y*y))+1) denotes the function x^2+y^2+1.

Attention! Note that the language of first order arithmetic **does not** contain a function constant denoting the **exponentiation** x^y, thus, for example, we must write x*x instead of x^{1+1}.

Of course, the key element of our efforts in describing "objects", their properties and relations, will be assertions, for example, the commutative law in arithmetic: ((x+y)=(y+x)). In predicate languages, assertions are called **formulas** (or, sometimes, well formed formulas – wff-s, sentences, or statements).

Atomic formulas

(In some other textbooks: elementary formulas, prime formulas.) Atomic formulas are defined as follows:

a) If p is a k-argument predicate constant, and t_1, ..., t_k are terms, then the

string $p(t_1, ..., t_k)$ is an atomic formula.

b) There are no other atomic formulas.

For the equality symbol, the "infix" notation is used: if t_1, t_2 are terms, then $(t_1=t_2)$ is an atomic formula.

Examples. 1) In our "language about people", the following represent examples of atomic formulas: *Male(x), Female(Britney), Male(Britney)* (not all formulas that are well formed, must be true!), *Father(x, Britney), Mother(Britney, John), Married(x, y).*

2) Summary of the atomic formulas of the language of first order arithmetic: a) constants 0 and 1, and all variables are terms; b) if t_1 and t_2 are terms, then (t_1+t_2) and (t_1*t_2) also are terms; c) atomic formulas are built only as $(t_1=t_2)$, where t_1 and t_2 are terms.

3) In the language of set theory, there are only two kinds of atomic formulas: $x \in y$, and x=y (where x and y are arbitrary variables).

In the language of first order arithmetic, even by using the only available predicate constant "=", atomic formulas can express a lot of clever things, for example,

((x+0)=x); ((x+y)=(y+x)); ((x+(y+z))=((x+y)+z));
((x*0)=0); ((x*1)=x); ((x*y)=(y*x)); ((x*(y*z))=((x*y)*z));
(((x+y)*z)=((x*z)+(y*z))).

Exercise 1.2.1. As the next step, translate the following assertions into the language of first order arithmetic (do not use abbreviations!): 2*2=4, 2*2=5, $(x+y)^2 = x^2+2xy+y^2$.

Formulas

The following definition is common for all predicate languages. Each language is specific only by its set of language primitives.

To write more complicated assertions, we need compound formulas, built of atomic formulas by using a fixed set of **propositional connectives** and **quantifiers** (an invention due to G. Frege and C. S. Peirce). In this book, we will use the following set of symbols:

Implication symbol: B→C means "if B, then C", or "B implies C", or "C follows from B".

Conjunction symbol: $B \wedge C$ means "B and C".

Disjunction symbol: $B \vee C$ means "B, or C, or both".

Attention! Thus, our disjunction symbol means the so-called **non-exclusive "or"**. If B and C both are true, then $B \vee C$ is true as well. The so-called **exclusive OR** (programmers would wish to write it as $B\,xor\,C$) is used to denote “either B, or C, but not both”.

Negation symbol: $\neg B$ means "not B".

Universal quantifier: $\forall x\, B$ means "for all x, B".

Existential quantifier: $\exists x\, B$ means "there is x such that B".

The widely used equivalence connective ↔ can be derived in the following way: B↔C stands for $((B \to C) \wedge (C \to B))$. If you wish to use the exclusive OR ("either B, or C, but not both"), you can define $B\, xor\, C$ as $\neg(B \leftrightarrow C)$.

Attention! For programmers, conjunction, disjunction and negation are familiar "logical operations" – unlike the implication that is not used in "normal" programming languages. In programming, the so-called IF-statements, when compared to logic, mean a different thing: in the statement IF x=y THEN z:=17, the condition, x=y is, indeed, a formula, but the "consequence" z:=17 is not a formula – it is an executable statement. In logic, in B→C ("if B, then C"), B and C both are formulas.

Now, we can define the notion of **formula** of a predicate language as follows:

a) Atomic formulas are formulas.

b) If B and C are formulas, then $(B \to C), (B \wedge C), (B \vee C)$, and $(\neg B)$ also are formulas (B and C are called sub-formulas).

c) If B is a formula, and x is an object variable, then $(\forall x B)$ and $(\exists x B)$ also are formulas (B is called a sub-formula).

d) There are no other formulas.

Warning about omitting of parentheses! To make formulas easier for human reading, usually, some of the formally necessary parentheses are omitted according to the so-called *priority rules*. For example,

$B \vee C \to D \wedge F$ means $(B \vee C) \to (D \wedge F)$, and **not** $(B \vee (C \to D) \wedge F)$.

Thus, implication has lower priority than disjunction and conjunction. For full treatment of priority rules see the section "Omitting Parentheses" below.

Knowledge representation by means of predicate languages

Do not be surprised by the trivial character of the most of the assertions expressed in the formulas below. Even such trivial assertions must be submitted to computers, if we wish them to make conclusions and answer questions about people, natural numbers, or sets.

Examples. 1) In our "language about people", the following are examples of compound formulas:

$((Father(x, y)) \vee (Mother(x, y)))$	"x is a parent of y"

$(\forall x(\forall y((Father(x,y))\rightarrow(Male(x)))))$	"fathers are males" – could serve as an **axiom**. Looks **somewhat artificial**, but represents the most natural way of saying that "fathers are males" in a predicate language!
$(\forall x(\forall y((Mother(x,y))\rightarrow(\neg Male(x)))))$	"mothers are not males" – could be **derived** from the axioms formulated below.
$(\forall x(\exists y(Mother(y,x))))$	"each x has some y as a mother" – could serve as an **axiom.**
$(\forall x(Male(x)\vee Female(x)))$	What does it mean? It could serve as an **axiom.**
$(\forall x\neg(Male(x)\wedge Female(x)))$	What does it mean? It could serve as an **axiom.**
$\exists z(Father(x,z)\wedge(Father(z,y)\vee Mother(z,y)))$ "x is a grandfather of y"	
$\forall x(\forall y(\forall z((Mother(x,z)\wedge Mother(y,z))\rightarrow(x=y))))$ $\forall x(\forall y(\forall z((Father(x,z)\wedge Father(y,z))\rightarrow(x=y))))$ What does this mean? These formulas could serve as **axioms**. They look **somewhat artificial**, but represent the most natural way of saying "no more than one" in a predicate language!	

2) Some simple examples of compound formulas in the language of first order arithmetic.

Attention! Speaking strictly, predicate symbols "<", ">", "≤", "≥", "≠" etc. do not belong to the language of first order arithmetic. For example, x<y should be replaced by their full version of the kind $\exists u((((x+u)+1)=y))$.

$(\exists u(x=(u+u)))$	"x is an even number"
$(\exists u(((x+u)+1)=y))$	"x is less than y", or, x<y
$(0<y\wedge\exists u(x=(y*u)))$	"x is divisible by y". Speaking strictly, x<y must be replaced by $\exists u(((x+u)+1)=y)$.
$((1<x)\wedge(\neg(\forall y(\exists z(((y<x)\wedge(z<x))\wedge(x=(y*z)))))))$, formula *prime(x)*, "x is a prime number".	

$\forall w(\exists x((w<x)\wedge(prime(x))))$	"There are infinitely many prime numbers" (one of the first mathematical theorems, 6th century BC). Speaking strictly, w<x must be replaced by $\exists u((((w+u)+1)=x))$, and *prime(x)* must be replaced by the above long formula.
$\forall x\forall y(0<y\rightarrow\exists z\exists u(u<y\wedge x=y*z+u))$	What does it mean?

3) Some simple examples of compound formulas in the language of set theory:

$(\exists y(y\in x))$	"x is a non-empty set"
$(\forall z((z\in x)\rightarrow(z\in y)))$	"x is a subset of y", or x≤y
$((\forall z((z\in x)\leftrightarrow(z\in y)))\rightarrow(x=y))$	What does it mean? Will serve as an **axiom**.
$(\forall y(\forall z((y\in x)\wedge(z\in x))\rightarrow y=z))$	"x contains zero or one member"
$(\forall u((u\in x)\leftrightarrow((u\in y)\vee(u\in z))))$	"x is union of y and z", or $x=y\cup z$

Of course, once again, **having a predicate language is not enough** for expressing all of our knowledge formally, i.e., for communicating it to computers. Computers do not know in advance, for example, how to handle sexes. We must tell them how to handle these notions by introducing **axioms**. Thus, the above-mentioned formulas like as $Female(Britney)$,

$\forall x(Male(x)\vee Female(x))$, or $\forall x\forall y(Father(x,y)\rightarrow(Male(x)))$

will be absolutely necessary as **axioms** (facts and rules). As we will see later,

knowledge base = language+facts+rules+query processor,

or, in mathematics,

theory = language + axioms + logic,

i.e., in fact, **to formulate all of our knowledge formally, we must create theories.**

Exercise 1.2.2. Translate the following assertions into our "language about people":

"x is child of y";
"x is grand-mother of y"; "x is a grand-father"
"x is brother of x"; "x and y are sisters";
"x is cousin of y"; "x is nephew of y"; "x is uncle of y".

Exercise 1.2.3. Translate the following assertion to the language of first order

arithmetic:

"x and y do not have common divisors" (note: 1 is not counted as a divisor!);

" $\sqrt{2}$ is an irational number"; (**Attention!** $\neg\exists p\exists q(\sqrt{2}=\frac{p}{q})$, and $\exists x(x*x=2)$ are not correct solutions. Why?)

Exercise 1.2.4. Imagine an alternative language of arithmetic that does not contain function constants + and *, but contains predicates *sum(x, y, z)* and *prod(x, y, z)* instead (meaning x+y=z and x*y=z correspondingly). Translate the following assertions into this language:

$$x+0=x\,;\,x+y=y+x\,;\,x+(y+1)=(x+y)+1\,;\,(x+y)*z=(x*z)+(y*z)\quad .$$

Exercise 1.2.5. Try inventing your own predicate language. Prepare and do your own Exercise 1.2.2 for it.

Exercise 1.2.6 (optional). In computer science, one of the popular means of knowledge representation are the so-called **UML class diagrams** and **OCL** (UML − Unified Modeling Language, OCL − Object Constraint Language). The above diagram representing our “domain of people” is an example. In our “language of people”, put down as many **axioms** of the domain you can notice in the diagram. For example, “every person is either male, or female”, “all fathers are males”, “every person has exactly one mother”, “a person can marry no more than one person” etc.

Many-sorted Languages

Maybe, you have to describe two or more kinds of "objects" that you do not wish to reduce to "sub-kinds" of one kind of "objects" (for example, integer numbers and character strings). Then you may need introducing for each of your "domains" a separate kind ("sort") of object variables. In this way you arrive to the so-called **many-sorted predicate languages**. In such languages: a) a finite number of sorts can be introduced; b) each variable and each object constant must be assigned to some sort; b) for each function constant, each of its arguments must be assigned to some some sort, and function values must be assigned to a (single) sort; c) for each predicate constant, each of its argument must be assigned to some sort. In many-sorted predicate languages, the term and atomic formula definitions are somewhat more complicated: building of the term $f(t_1, ..., t_k)$ or the formula $p(t_1, ..., t_k)$ is allowed only, if the sort of the term t_i values coincides with the sort of the i-th argument of f or p respectively. And the "meaning" of quantifiers depends on the sort of the variable used with them. For example, $\forall x$ means "for all values of x from the domain of the sort of x".

Theoretically, many-sorted languages can be reduced to one-sorted languages by introducing the corresponding predicates $Sort_i(x)$ ("the value of x belongs to the sort i"). Still, in some applications of logic (for example, in computer science) the many-sorted approach is usually more natural and more convenient.

Warning about second order languages!

In our definition of predicate languages only the following kinds of primitives were used: object variables, object constants, function constants and predicate constants. You may ask: how about **function variables** and **predicate variables**? For, you may wish to denote by r "an arbitrary property" of your "objects". Then, r(x) would mean "x possess the property r", and you would be able to say something about "all properties", for example, $\forall r \forall x \forall y (x = y \rightarrow r(x) \leftrightarrow r(y))$. In this way you would have arrived at a **second order language**! In such languages, function and predicate variables are allowed. But properties lead to sets of objects, for example, {x | r(x)} would mean the set of all objects that possess the property r. But, why should we stop at the properties of objects? How about "properties of sets of objects" etc.? As it was detected long ago, all kinds of sets can be fully treated only in **set theory**! Thus, instead of building your own second order language, you should better try applying your favorite ("first order") set theory. An unpleasant consequence: the existence of the (much less significant) notion of second order languages forces many people to call predicate languages "**first order languages**" − to emphasize that, in these languages, the only kind of variables allowed are object variables.

On the other hand, when trying to implement realistic **formal reasoning software**, then using of some second order constructs is, as a rule, more efficient than implementing of a pure first order reasoning. See, for example, *Notices of the AMS*, Special Issue on Formal Proof, Vol. 55, N 11, 2008 (available online).

For details, see: Second-order-logic in Wikipedia.

Omitting parentheses

Our formal definitions of terms and formulas lead to expressions containing many parentheses. Let us remind, for example, our formula expressing that "x is a prime number":

$$((1<x)\wedge(\neg(\exists y(\exists z(((y<x)\wedge(z<x))\wedge(x=(y*z)))))))\quad .$$

Such formulas are an easy reading for computers, yet inconvenient for human reading (and even more inconvenient – for writing them correctly). In the usual mathematical practice (and in programming languages) we are allowed to improve the look of our formulas by omitting some of the parentheses − according to (some of) the following rules:

a) Omit the outermost parentheses, for example, we may write A→(B→C) instead of the formally correct (A→(B→C)). In this way we may improve the final look of our formulas. Still, if we wish to use such formulas as parts of more complicated formulas, we must restore the outermost parentheses, for example: (A→(B→C))→D.

b) We may write, for example, simply:

$$x+y+z+u\,,\; x*y*z*u\,,$$
$$A\wedge B\wedge C\wedge D\,,\; A\vee B\vee C\vee D\,,$$
$$\exists x \forall y \exists z \forall u F\,.$$

instead of the more formal

$$((x+y)+z)+u, ((x*y)*z)*u,$$
$$((A \wedge B) \wedge C) \wedge D,$$
$$((A \vee B) \vee C) \vee D,$$
$$(\exists x(\forall y(\exists z(\forall u F)))).$$

In this way we can simplify the above expression "x is a prime number" as:

$$(1<x) \wedge (\neg(\exists y \exists z((y<x) \wedge (z<x) \wedge (x=(y*z))))) \quad .$$

c) We can apply the so-called **priority rules**. For example, the priority rank of multiplications is supposed to be higher than the priority rank of additions. This rule allows writing x+y*z instead of the more formal x+(y*z) − because of its higher priority rank, multiplication must be performed first. The most popular priority rules are the following:

c1) The priority rank of **function constants** is higher than the priority rank of **predicate constants**. This allows, for example, writing x*y = y*x instead of (x*y)=(y*x), or $x \in y \cup z$ − instead of $x \in (y \cup z)$.

c2) The priority rank of **predicate constants** is higher than the priority rank of **propositional connectives and quantifiers**. This allows, for example, writing $y<x \wedge z<x$ instead of $(y<x) \wedge (z<x)$.

c3) The priority rank of **quantifiers** is higher than the priority rank of **propositional connectives**. This allows, for example, writing $\exists x F \wedge \forall y G$ instead of $(\exists x(F)) \wedge (\forall y(G))$, or writing $\neg \exists x F$ instead of $\neg(\exists x(F))$.

c4) The priority rank of **negations** is higher than the priority rank of **conjunctions and disjunctions**. This allows, for example, writing $\neg A \wedge \neg B$ instead of $(\neg A) \wedge (\neg B)$.

c5) The priority rank of **conjunctions and disjunctions** is higher than the priority rank of **implications**. This allows, for example, writing $A \wedge B \rightarrow C \vee D$ instead of $(A \wedge B) \rightarrow (C \vee D)$.

In the usual mathematical practice, some additional priority rules are used, but some of them are not allowed in the common programming languages. To avoid confusions do not use too many priority rules simultaneously!

According to the above priority rules, we can simplify the above expression "x is a prime number" even further, obtaining a form that is much easier for human reading (but is somewhat complicated for a computer program to “parse” it):

$$1<x \wedge \neg \exists y \exists z(y<x \wedge z<x \wedge x=y*z) \quad .$$

As you see, all the above rules are mere abbreviations. In principle, you could use any other set of abbreviation rules accepted by your audience. If

computers would do logic themselves, they would not need such rules at all (except, maybe, for displaying some of their results to humans).

Exercise 1.2.6. "Translate" the following assertions to our "language about people":

"x and y are siblings";
"x and y are brothers"; “x and y are sisters”; “x is cousin of y”;
“parents of x and y are married”;
“x and y are married people” (formalize both possible meanings);
“nobody is married to everybody”.

Exercise 1.2.7. Translate the following assertions to the language of first order arithmetic:

"x and y are twin primes" (examples of twin pairs: 3,5; 5,7; 11,13; 17,19;...),
"There are infinitely many pairs of twin primes" (the famous **Twin Prime Conjecture**),
"Each positive even integer ≥4 can be expressed as a sum of two primes" (the famous **Goldbach Conjecture**),

"x is a power of 2" (**Attention!** $\exists n(x=2^n)$ is not a correct solution. Why? Think about prime divisors of x instead.).

Free variables and bound variables

Usually, people use these notions without definitions. Anyway, a precise treatment follows.

The above expression "x is a prime number":

$$1<x\wedge\neg\exists y\exists z\left(y<x\wedge z<x\wedge x=y*z\right)$$

contains 3 variables: x − occurs 4 times in terms, y − 2 times in terms and 1 time in quantifiers, z − occurs 2 times in terms and 1 time in quantifiers. Of course, x is here a "free" variable – in the sense that the "truth value" of the formula depends on particular "values" taken by x. On the contrary, the "truth value" of the formula does not depend on the particular "values" taken by the two "bound" variables y and z − the quantifiers ∃y, ∃z force these variables to "run across their entire range".

To make the notion precise, first, let count only the occurrences of variables in *terms*, not in quantifiers. And second, let us define a particular occurrence o_x of a variable x in (a term of) a formula F as a **free occurrence** or a **bound occurrence** according to the following rules:

a) If F does not contain quantifiers ∃x, ∀x, then o_x is free in F.

b) If F is ∃xG or ∀xG, then o_x is bound in F.

c_1) If F is $G \wedge H$, $G \vee H$, or G→H, and o_x is free in G (or in H), then o_x is free in F.

c_2) If F is ¬G, ∃yG, or ∀yG, where y **is not** x, and o_x is free in G, then o_x is free in F.

d_1) If F is $G \wedge H$, $G \vee H$, or G→H, and o_x is bound in G (or in H), then o_x is bound in F.

d_2) If F is ¬G, ∃yG, or ∀yG (where y is any variable, x included), and o_x is bound in G, then o_x is bound in F.

Thus, the above formula $1<x \wedge \neg \exists y \exists z (y<x \wedge z<x \wedge x=y*z)$ contains 4 free occurrences of x, 2 bound occurrences of y, and 2 bound occurrences of z.

Exercise 1.2.8. Verify that an occurrence of x in F cannot be free and bound simultaneously. (Hint: assume that it is not the case, and consider the sequence of all sub-formulas of F containing this particular occurrence of x.)

Formally, we can use formulas containing free and bound occurrences of a single variable simultaneously, for example, $x>1 \rightarrow \exists x (x>1)$. Or, many bound occurrences of a single variable, for example,

$$(\forall x F(x) \wedge \exists x G(x)) \vee \forall x H(x)$$

means the same as

$$(\forall x F(x) \wedge \exists y G(y)) \vee \forall z H(z) \quad .$$

Still, it is not recommended using a single variable in too many different roles in a single formula. Such formulas do not cause problems for computers, but they may become inconvenient for human reading.

If a formula contains free occurrences of variables, i.e., occurrences of variables that are not bound by quantifiers (for example: $x=0 \vee x=1$), then the "truth value" of such formulas may depend on particular values assigned to free variables. For example, the latter formula is "true" for x=1, yet it is "false" for x=2. Formulas that do not contain free occurrences of variables, are called **closed formulas**, for example:

$$\forall w \exists x (w<x \wedge prime(x)) \quad .$$

Closed formulas represent "definite assertions about objects of theory", they are expected to be (but not always really are) either "true", or "false".

Term substitution

To say that x is a free variable of the formula F, we may wish to write F(x) instead of simply F. Replacing all free occurrences of x by a term t yields an

"instance" of the formula F. It would be natural to denote this "instance" by F(t).

For example, if F(x) is ∃y(y+y=x) and t is z*z+z, then F(t), or F(z*z+z) will denote ∃y(y+y=z*z+z).

However, if t would be y*y+y, then F(t), or F(y*y+y) would be ∃y(y+y=y*y+y). Is this really F(y*y+y)?

Thus, sometimes, formal substitutions can lead to crazy results. Another example: in our expression "x is a prime number", let us replace x by y. Will the resulting formula mean "y is a prime number"? Let's see:

$$1<y\wedge\neg\exists y\exists z\left(y<y\wedge z<y\wedge y=y*z\right)\ .$$

Since y<y is always false, the second part ¬∃y∃z(...) is true, hence, the latter formula means simply that "1 is less than y", and not that "y is a prime number".

Of course, we failed because we replaced a free variable x by a variable y in such a way that some **free** occurrence of x became **bound** by a quantifiers for y. In this way we **deformed the initial meaning** of our formula.

The following simple rule allows to avoid such situations. Suppose, x is a free variable of the formula F. We will say that the **substitution F(x/t)** (i.e., the substitution of the term t for x in the formula F) **is admissible** if and only if no free occurrences of x in F are located under quantifiers that bind variables contained in t. If the substitution F(x/t) is admissible, then, by replacing all free occurrences of x in F by t, of course, we do not change the initial meaning of the formula F(x), and hence, we may safely denote the result of this substitution by F(t).

Exercise 1.2.9. Is x/y an admissible substitution in the following formulas? Why?

$$x=0\vee\exists y\left(y>z\right)\ ;$$
$$x=0\vee\exists y\left(y>x\right)\ .$$

Exercise 1.2.10 (optional). a) Mathematicians: think over the analogy between bound variables in logic and bound variables in sum expressions and integrals.
b) Programmers: think over the analogy between bound variables in logic and loop counters in programs.

1.3. Axioms of Logic: Minimal System, Constructive System and Classical System

Now we go on in detail to the second phase of formalization:

a) after having defined a *formal language* (predicate language) allowing to put down propositions about objects in our domain of interest,

b) and having formulated as *axioms* some of the propositions, that we think to be "true" of the objects in the domain,

c) we must introduce some *means of reasoning* allowing to derive other statements that are as "true" of the objects as are the axioms.

Indeed, having formulated some fragment of our knowledge as a set of axioms A_1, ..., A_n in some predicate language L, and put them into a knowledge base, we do not think that A_1, ..., A_n are **all** the statements that are "true" of the objects we are trying to describe. Many other "true" statements **follow** from A_1, ..., A_n as consequences.

The problem of reasoning:

"formula G follows from the formulas A_1, ..., A_n",

what exactly does it mean? Since we wish to **teach reasoning to computers**, the answer must be absolutely explicit.

Tentative solution of the problem

Axioms of a theory can formulate facts and rules. **Facts** are formulated usually as atomic formulas that do not contain variables, for example,

$$Male(John);$$
$$Female(Britney).$$

Rules are formulated as formulas that contain logical connectives and quantifiers, for example,

$$\forall x(Male(x) \vee Female(x));$$
$$\neg \exists x(Male(x) \wedge Female(x));$$
$$\forall x \forall y(Father(x, y) \rightarrow Male(x)).$$

Thus, to teach reasoning to computers, we must, first of all, teach them **how to manipulate logical connectives and quantifiers**. The necessary principles of manipulation will be represented as **logical axioms and rules of inference**. Since the "meaning" of connectives and quantifiers does not depend on the

"meaning" of the specific primitives of each predicate language, these axioms and rules of inference must be applicable to any predicate languages. This is because, sometimes, such axioms and rules are called "generally valid", "logically valid", or "purely logical".

For example, assume that some formula F has the following form:

$$(B \to D) \to ((C \to D) \to (B \vee C \to D)) \quad ,$$

where B, C, D are some formulas. Then F is "true" independently of the specific language primitives used in the formulas B, C, D. This is why it is accepted below as the logical axiom $\mathbf{L_8}$.

Similarly, the following rule of inference (*Modus Ponens*) is applicable independently of the language primitives contained in the formulas B, C:

Having derived the formulas B, B→C, derive the formula C.

For example, if we have B→D and C→D already derived, then – by applying twice this rule to the above long formula F – we can derive that $B \vee C \to D$.

We will try to formulate a complete (as complete as possible) system of such "purely logical" principles (logical axioms and rules of inference). Establishing the existence of such a system is the result of a 2500 year long history of great discoveries and inventions.

Aristotle (384-322 BC),

Gottlob Frege (1848-1925), Charles Sanders Peirce (1839-1914).

Bertrand Russell (1872-1970), David Hilbert (1862-1943).

D.Hilbert, W.Ackermann. Grundzüge der theoretischen Logik. Berlin (Springer), 1928 (see also: Hilbert and Ackermann's 1928 Logic Book by Stanley N. Burris).

The first version of logical axioms was introduced in 1879 by G. Frege in his above-mentioned *Begriffsschrift*. The next important version was proposed in 1910-1913 by B. Russell and A. Whitehead in their famous book *Principia Mathematica*. And finally, in 1928 D. Hilbert and W. Ackermann published in their above-mentioned book, in a sense, the final version of logical axioms. Equivalent modifications of this version are now used in all textbooks of mathematical logic.

In our version, logical axioms will be represented by means of the so-called **axiom schemas** (programmers might call them *templates*). Each schema (template) represents an infinite, yet easily recognizable collection of single axioms. For example, schema L_3: $B \wedge C \to B$ may represent the following axioms ("instances of the schema") in the language of first order arithmetic:

$$x = y \wedge x = x \to x = y \quad ,$$

$$1 * 1 = 1 \wedge 1 + 1 = 1 + 1 \to 1 * 1 = 1 \quad ,$$

and many other axioms: take any formulas B, C in your predicate language, and you will obtain an instance of the axiom schema $B \wedge C \to B$.

We will not specify properties of the **equivalence** connective in axioms. We will regard this connective as a derived one (as a "macro"): $B \leftrightarrow C$ will be used as an abbreviation of $(B \to C) \wedge (C \to B)$. Similarly, $A \; xor \; B$ ("exclusive OR") can be used as an abbreviation of $\neg(B \leftrightarrow C)$.

Axioms of logic

Suppose, we have specified some predicate language L. We adopt the following 15 axiom schemas as **the logical axioms for the language L.**

In the first 11 axiom schemas L_1-L_{11} below, B, C and D are any formulas in the language L.

The first two axiom schemas L_1, L_2 represent the "definition" of the implication connective:

L_1: $B \to (C \to B)$ (try thinking over, what does it mean?),

L_2: $(B \to (C \to D)) \to ((B \to C) \to (B \to D))$ (think over, what does it mean?).

Such (or similar) definition of implication is necessary, if we wish computers to handle implications correctly.

Note. The axioms L_1, L_2 represent the (currently) most popular version of "defining" the implication connective. About other (equivalent) versions − containing 3 or 4 axioms − see Hilbert, Bernays [1934] (Chapter III) and Exercise 1.5.2 below.

The following axiom schemas L_3–L_5 represent the "definition" of the AND-connective (conjunction):

L_3: $B \wedge C \to B$ (what does it mean?),

L_4: $B \wedge C \to C$ (what does it mean?),

L_5: $B \to (C \to B \wedge C)$ (what does it mean?).

Such (or similar) definition of conjunction is necessary, if we wish computers to handle conjunctions correctly.

The following axiom schemas L_6–L_8 represent the "definition" of the (**non-exclusive!**) OR-connective (disjunction):

L_6: $B \to B \vee C$ (what does it mean?),

L_7: $C \to B \vee C$ (what does it mean?),

L_8: $(B \to D) \to ((C \to D) \to (B \vee C \to D))$ (what does it mean?).

Such (or similar) definition of disjunction is necessary, if we wish computers to handle disjunctions correctly.

Attention! Once again, as the above three axioms show, the disjunction connective is intended as the **non-exclusive OR**: $B \vee C$ is meant as “B, or C, or both”.

The next axiom schema L_9 represents the "definition" of the negation connective. In fact, it is a formal version of a proof method well-known in mathematics − refutation by deriving a contradiction (*Reductio ad absurdum*):

L_9: $(B \to C) \to ((B \to \neg C) \to \neg B)$ (what does it mean?).

Note. The axiom L_9 represents the (currently) most popular version of "defining" the negation connective. About other (equivalent) versions − see Hilbert, Bernays [1934] (Chapter III) and Exercise 2.4.2 below.

The next axiom schema L_{10} represents the famous principle "Contradiction Implies Anything" (*Ex contradictione sequitur quodlibet*, or *Ex falso sequitur quodlibet*):

L_{10}: $\neg B \to (B \to C)$ (try thinking over, what does it mean?).

The following axiom schema L_{11} represents the famous **Law of Excluded Middle** (*Tertium non datur)*:

L_{11}: $B \vee \neg B$ (try thinking over, what does it mean?).

The above 11 schemas (plus the *Modus Ponens* rule of inference, see below) represent the **classical propositional logic** in the language L.

Now, the "definitions" of the universal and existential quantifiers follow.

In the following axiom schemas L_{12}, L_{13}, F is any formula, and *t* is a term such that F(x/t) is an admissible substitution (in particular, *t* may be *x* itself):

L_{12}: $\forall x F(x) \to F(t)$ (in particular, $\forall x F(x) \to F(x)$, what does it mean?),

L_{13}: $F(t) \to \exists x F(x)$ (in particular, $F(x) \to \exists x F(x)$, what does it mean?).

In the following schemas L_{14}, L_{15}, F is any formula, and G is a formula that does not contain free occurrences of the variable *x*:

L_{14}: $\forall x (G \to F(x)) \to (G \to \forall x F(x))$ (what does it mean?),

L_{15}: $\forall x (F(x) \to G) \to (\exists x F(x) \to G)$ (what does it mean?).

Rules of inference

In the following rules of inference, B, C and F are any formulas.

***Modus Ponens* (MP)**: B→C; B ⊢ C, or, $\frac{B \to C \; ; \; B}{C}$ (what could this mean?).

Generalization (Gen): $\frac{F(x)}{\forall x \, F(x)}$ (what could this mean?).

This list of logical axioms and rules of inference represents the so-called **classical predicate logic** in the predicate language L (or, simply − the **classical logic** in the language L).

Note. In the above axioms, properties of the **equality predicate** x=y (such as reflexivity, symmetry, transitivity etc.) are not defined. So, equality is considered here as a **non-logical symbol**. As a rule, the concept of equality is specific in each particular theory. For example, in set theories, two sets are considered as equal, if and only of they possess the same members, i.e., x=y means $\forall z(z \in x \leftrightarrow z \in y)$. The necessary properties of equality must be formulated in the specific axioms of theory, or must be provable by using these axioms. This is why, in some texts, the above axioms of logic are called **first order logic without equality**. For an alternative version of logic, in which equality is considered as a logical symbol, see First order logic with equality in Wikipedia.

Possible misconceptions

1. *Modus Ponens* rule allows to derive C, if we have B and B→C (C follows from B) already derived. The "converse" version B→C; C ⊢ B is **wrong as a logical principle**. Knowing B→C and C can serve as an argument in favor of B being true, but not as a reliable *proof* of B.

2. No formal restrictions are put on the formula *F(x)* when applying *Generalization* rule. However, be careful when using *Gen* in a proof: the result $\forall x \, F(x)$ will be valid only if *F(x)* was derived putting no restrictions on *x*. If our proof contains such restrictions, i.e., if F(x) was derived by using some hypothesis *H(x)*, then, from *F(x)*, we cannot conclude $\forall x \, F(x)$. If one is applying *Gen* to F(x), one has in mind that F(x) is "true for all values of x". For details, see Deduction Theorem 2 in Section 1.5 below.

Some of the logical axioms are "wrong, but useful"!

Three of the above axiom schemas seem to be (at least partly) problematic.

For example, how do you find the funny axiom L_{10}: $\neg B \to (B \to C)$? If ¬B and B were true simultaneously, then anything were true? *Ex contradictione sequitur quodlibet?* Is this a really "true" axiom? Of course, it is not. Still, this does not matter: we do not need to know, were C "true" or not, if ¬B and B

were "true" simultaneously. By assuming that "if ¬B and B were true simultaneously, then anything were true" **we greatly simplify our logical apparatus**. For example, we will prove in Section 2.6 that, in the classical logic, ¬¬B→B. This simple formula can't be proved without the "crazy" axiom L_{10} (see Section 2.8).

In fact, the first axiom L_1: B→(C→B) is funny as well. If B is (unconditionally) true, then B follows from C, even if C has nothing in common with B? Moreover, in Exercise 1.4.2 below we will prove that the axioms L_1, L_9 allow proving that ¬B, B ⊢ ¬C, i.e., if ¬B and B were true simultaneously, then anything were false (thus, in a sense, L_1 contains already 50% of L_{10}!). After this, could we think of L_1 as a really "true" axiom? Of course, we can't. Still, this does not matter: if B is already known as true, then it may be too complicated to explore all C's from which B could follow. By assuming that "if B is true, then B follows from anything" **we greatly simplify our logical apparatus**.

The above two phenomena are called **paradoxes of the material implication**, see Paradoxes of Material Implication by Peter Suber, and Falsity Implies Anything by Alexander Bogomolny.

May our decision to "greatly simplify" the logical apparatus have also some undesirable consequences? Let us consider the following formula F(x): $\forall y(Child(x,y) \to Female(y))$. It seems, F(x) is intended to mean: "All the children of *x* are female". However, in our system of logic, F(x) is regarded as true also, if *x* does not have children at all! If you do not have children at all, then all your children are female! Or male? Or smart? Etc. Sounds funny, but is, in fact, harmless...

Thus, it would be an exaggeration to call the above system of axioms and rules of inference **"a system of principles of correct reasoning"**. As we saw it, not all logical axioms represent such principles, some of them were introduced solely in order to simplify the system. It would be better to call this system **"a good engine of reasoning".** Though not perfect, it is extremely efficient, much more efficient than the more complicated systems proposed in order to overcome its seeming deficiencies (see, for example, Relevance logic in Wikipedia).

Constructive logic

Still, it appears that the most serious problem is caused not by L_1 and L_{10}, it is caused by the axiom L_{11}: $B \vee \neg B$ – the Law of Excluded Middle. How can we think of L_{11} as a "true" axiom, if (according to Gödel's Incompleteness

Theorem) each sufficiently strong consistent theory contains undecidable propositions? We postulate that either B, or ¬B "must be true", yet for some B we will unable to prove neither B, nor ¬B! Knowing that $B \vee \neg B$ is "true" inspires us to work on the problem, but it may appear useless, if we do not succeed... Should we retain L_{11} as an axiom after this?

For this (and some other, see below) reasons some people reject L_{11} as a "valid" logical axiom.

The above list of 15 axiom schemas as it stands is called the **classical logic**.

By excluding L_{11} from the list we obtain the so-called **constructive** (historically, and in most textbooks − **intuitionistic**) **logic**. As a concept, it was introduced by Luitzen Egbertus Jan Brouwer in 1908:

L. E. J. Brouwer. De onbetrouwbaarheid der logische principes (The unreliability of the logical principles), *Tijdschrift voor Wijsbegeerte*, 2 (1908), pp.152-158.

Brouwer's main objection was against non-constructive proofs that are enabled mainly by an "improper" use of the Law of Excluded Middle.

An elegant very short *non-constructive* proof proposes to use either the pair $a=b=\sqrt{2}$, or the pair $a=\sqrt{2}; b=\sqrt{2}^{\sqrt{2}}$ to prove that there are two irrational numbers *a, b* such that a^b is rational. The possibly shortest *constructive* proof of this fact is by using the pair $a=\sqrt{2}; b=2\log_2 3$ instead.

Exercise 1.3.1. Elaborate on these two proofs.

For the entire history, see The Root-2 Proof as an Example of Non-constructivity by J. Roger Hindley.

Note. A similar kind of non-constructive reasoning is represented by the so-called Double Negation Law: ¬¬B→B, see Section 2.6.

As a formal system, the intuitionistic logic was formulated by Arend Heyting in 1930:

A. Heyting. Die formalen Regeln der intuitionistischen Mathematik. *Sitzungsberichte der Preussischen Akademie der Wissenschaften, Physikalisch-mathematische Klasse*, 1930, pp.42-56.

The constructive concept of logic differs from the classical one mainly in its interpretation of disjunction and existence assertions:

− To prove $B \vee C$ **constructively**, you must prove B, or prove C. To prove $B \vee C$ by using the *classical* logic, you are allowed to assume $\neg(B \vee C)$ as a hypothesis and derive a contradiction. Having only such a "negative" proof, you may be unable to determine, which part of the disjunction $B \vee C$ is true − B, or C, or both. Knowing that $B \vee C$ is "true" may inspire you to work on the problem, but it may appear useless, if you do not succeed…

− To prove $\exists x B(x)$ **constructively**, you must provide a particular value of x such that B(x) is true. To prove $\exists x B(x)$ by using the classical logic, you are allowed to assume $\forall x \neg B(x)$ as a hypothesis to derive a contradiction. Having only such a "negative" proof, you may be unable to find a particular x for which B(x) is true. Knowing that $\exists x B(x)$ is "true" may inspire you to work on the problem, but it may appear useless, if you do not succeed...

Note. As a joke, we could regard existence assertions as "huge disjunctions". For example, in the language of first order arithmetic, $\exists x B(x)$ could be "thought" of as $B(0) \vee B(1) \vee B(2) \vee \ldots$, i.e., as an infinite "formula". Thus, the above two theses are, in a sense, "equivalent".

The constructive (intuitionist) logic is one of the great discoveries in mathematical logic − surprisingly, a system of constructive reasoning can be obtained simply by dropping the Law of Excluded Middle from the list of valid logical principles.

See also Intuitionistic Logic by Joan Moschovakis in Stanford Encyclopedia of Philosophy.

See also on Markov's Principle in Wikipedia.

Exercise 1.3.2. Explain, why the following formulas cannot be proved in the constructive logic:

a) $\neg\neg B \to B$;

b) $\neg \forall F(x) \to \exists x \neg F(x)$;

c) $\neg(B \wedge C) \to \neg B \vee \neg C$.

Minimal logic

By excluding both L_{10} and L_{11} we obtain the so-called **minimal logic**. It was introduced by Ingebrigt Johansson in 1936:

I.Johansson. Der Minimalkalkül, ein reduzierter intuitionistischer Formalismus. *Compositio Mathematica*, 1936, Vol. 4, N1, pp.119-136.

As a separate concept, the minimal logic is much less significant than the constructive logic. Indeed, since it allows proving of [L_1, L_9, MP]: B, ¬B ⊢ ¬C (in a sense, 50% of L_{10}!), dropping of L_{10} is not a very big step.

Exercise 1.3.3 (optional). Reconsider Exercise 1.1.7 and verify that, for any predicate language, **minimal, constructive and classical predicate logic are formal theories** (with empty sets of non-logical axioms) according to the definition given in Section 1.1.

First order theories

Thus, as the result of the formalization process, we obtain the so-called **first order theories**.

Each first order theory T includes:

a) a specific predicate **language** L(T);

b) **logical axioms and rules of inference** for this language (classical or constructive version may be adopted);

c) a set of **specific (non-logical) axioms** of T (the specific knowledge, represented in T, the specific knowledge stored in the knowledge base).

Specific axioms are called "non-logical" because they provide information about the intended object domain of the theory. As such, they cannot be derived from logical axioms that are valid independently of any domains. See the examples below.

As we will prove in Section 4.3, **we will never need to introduce *specific (non-logical) rules of inference*.** All the consequences of the axioms of a first order theory, can be derived by using the logical axioms and two rules of inference – *Modus Ponens* and *Generalization*.

Thus, we have arrived at a tentative solution to the **problem of reasoning**: we have now a precise answer to the question stated above: "formula G follows from A_1, ..., A_n", what does it mean? It means: there is a correct proof (a sequence of formulas) that proves (the notation is explained below):

[L_1-L_{15}, MP, Gen]: $A_1, A_2, ..., A_n \vdash G$ (if we intend to use the classical logic), or proves

[L_1-L_{10}, L_{12}-L_{15}, MP, Gen]: $A_1, A_2, ..., A_n \vdash G$ (if we intend to use the constructive logic).

Is this solution, indeed, only a tentative one? In Section 4.3 we will prove Gödel's Completeness Theorem showing that, at least for the classical predicate logic, this solution is, in a sense, the only possible.

As the first example, let's use our "language about people" to build a "theory about people".

First of all, this theory includes instances of logical axioms for the "language about people", for example:

L_1: $Male(x) \rightarrow (Female(x) \rightarrow Male(x))$;

L_6: $Mother(x, y) \rightarrow Mother(x, y) \vee Father(x, y)$;

L_{11}: $Male(John) \vee \neg Male(John)$;

L_{13}: $Female(Britney) \rightarrow \exists x\, Female(x)$.

As we see here, the **logical axioms are "content-free"** – being applicable to

any languages, they cannot provide specific information about John, Britney, sexes and parents. This information must be provided by non-logical axioms.

This is why we must include as many as possible non-logical axioms, expressing what we think is "true" of our intended domain, for example:

$\forall x(Male(x)\vee Female(x))$;
$\neg\exists x(Male(x)\wedge Female(x))$;
$\forall x\forall y(Father(x,y)\rightarrow Male(x))$;
$\forall x\forall y\forall z((Mother(x,z)\wedge Mother(y,z))\rightarrow x=y)$.

Exercise 1.3.4. Extend this list of axioms as far as you can. Is your list complete? What do you mean by "complete"?

Another example of a first order theory − the so-called **first order arithmetic PA** (also called Peano arithmetic):

The language of PA:

a) Constants 0 and 1, and all variables are terms.

b) If t_1 and t_2 are terms, then (t_1+t_2) and (t_1*t_2) also are terms.

c) Atomic formulas are built as $(t_1=t_2)$, where t_1 and t_2 are terms.

Since we can use, for example, the expression $2x^2-3y^2-1=0$ as an abbreviation of (1+1)*x*x=(1+1+1)*y*y+1, we can say simply that, in first order arithmetic, atomic formulas represent Diophantine equations.

Examples of instances of logical axioms for the language of first order arithmetic:
L_1: $x=0\rightarrow(y=1\rightarrow x=0)$;
L_6: $x=y\rightarrow x=y\vee z=1$;
L_{11}: $0=1\vee\neg(0=1)$;
L_{12}: $\forall x(x=1)\rightarrow x=1$.

Once again, we see here, that the **logical axioms are "content-free"** – being applicable to any languages, they cannot provide specific information about natural numbers.

The specific (non-logical) axioms of first order arithmetic:

x=x,
x=y→y=x,
x=y→(y=z→x=z),
x=y→x+1=y+1,
¬(0=x+1),
x+1=y+1→x=y,
x+0=x,
x+(y+1)=(x+y)+1,
x*0=0,
x*(y+1)=(x*y)+x,
$B(0)\wedge\forall x(B(x)\rightarrow B(x+1))\rightarrow\forall x\,B(x)$, where B is any formula in the language of arithmetic.

The axioms 7-10 represent recursive definitions of addition and multiplication. As the last the

so-called **induction schema** is listed.

Note. But how about the exponentiation? Why didn't we include the function x^y in the language of arithmetic (and the corresponding axioms: $x^0=1$; $x^{y+1}=(x^y)*x$)? This is not necessary, because it appears that exponentiation can be defined as a (very complicated) formula containing only addition and multiplication. See Mendelson [1997], or Podnieks [1997], Section 3.3.

For the most popular **axiom system of set theory** – see ZFC (see Zermelo-Fraenkel's set theory in Wikipedia).

Proofs and theorems

In general, any sequence of formulas F_1, F_2, ..., F_m could be regarded as a (correct or incorrect) **formal proof** (or simply, a proof) of its last formula F_m. In a **correct proof**, formulas can play only the following roles:

a) *Axioms*. Some formulas may be logical or non-logical axioms (or instances of the corresponding schemas).

b) *Consequences* of earlier formulas, obtained by using the rules of inference. For example, if F_{25} is A, and F_{34} is A→B, and F_{51} is B, then we can say that F_{51} has been obtained from F_{25} and F_{34} by using the *Modus Ponens* rule. Or, if F_{62} is C(x), and F_{63} is $\forall$xC(x), then we can say that F_{63} has been obtained from F_{62} by using the *Generalization* rule.

c) *Hypotheses*. Some formulas may appear in the proof without any formal justification, simply by assuming that (for this proof) they are "true".

Thus, to describe the actual status of a formal proof, we need the following notation:

$$[T]: A_1, A_2, ..., A_n \vdash F,$$

where T is a first order theory (it determines which formulas are axioms and which are not), A_1, A_2, ..., A_n are **all** the hypotheses used in the proof, and F is the formula proved by the proof. Each formula in such a proof must be either an axiom, or a hypothesis from the set A_1, A_2, ..., A_n, or it must be obtained from earlier formulas (in this proof) by using a rule of inference. You may read the above notation as "in theory T, by using formulas A_1, A_2, ..., A_n as hypotheses, the formula F is proved".

As the first example, let us consider the following proof:

$$[L_5, MP]: \quad B, C \vdash B \wedge C \quad .$$

(1)	B	Hypothesis given.
(2)	C	Hypothesis given.
(3)	$B \to (C \to B \wedge C)$	It's the axiom schema L_5.
(4)	$C \to B \wedge C$	It follows from (1) and (3) by *Modus Ponens*.
(5)	$B \wedge C$	It follows from (2) and (4) by *Modus Ponens*.

For more serious examples of formal proofs see the next Section 1.4 (Theorem 1.4.1 and Theorem 1.4.2).

Using previously proved formulas

In practice, when proving [T]: $A_1, A_2, ..., A_n \vdash B$, we may wish to apply some theorem Q that already has been proved earlier. If we would simply insert Q into our formal proof, then, formally, this would yield only a proof of [T]: $A_1, A_2, ..., A_n, Q \vdash B$, i.e., we will be forced to qualify Q as a hypothesis. To obtain the desired **formal** proof of [T]: $A_1, A_2, ..., A_n \vdash B$, we must insert not only Q itself, but **the entire proof** of Q! In this way we obtain the following

Theorem 1.3.1. If there is a proof [T]: $A_1, A_2, ..., A_n, Q \vdash B$, and a proof [T]: $A_1, A_2, ..., A_n \vdash Q$, then there is a proof [T]: $A_1, A_2, ..., A_n \vdash B$.

Proof. The proof [T]: $A_1, A_2, ..., A_n, Q \vdash B$ is a sequence of formulas $F_1, F_2, ..., F_k, Q, ..., F_m, B$, and the proof [T]: $A_1, A_2, ..., A_n \vdash Q$ is some sequence of formulas $G_1, G_2, ..., G_p, Q$. Let us replace Q by $G_1, G_2, ..., G_p, Q$:

$$F_1, F_2, ..., F_k, G_1, G_2, ..., G_p, Q, ..., F_m, B,$$

and (if we wish so) eliminate the duplicate formulas. This sequence is a proof [T]: $A_1, A_2, ..., A_n \vdash B$. Q.E.D.

In practice, we are not interested in writing down full formal proofs in the sense of the above definition, where only axioms and hypotheses were allowed in proofs. Theorem 1.3.1 allows to **assert the existence of such proofs** without writing them down fully.

If, in some proof, hypotheses are not used at all, then we may write simply [T]: $\vdash B$, or even $T \vdash B$, and say that B is a **theorem** of theory T.

Corollary 1.3.1. If there is a proof [T]: A_1, A_2, ..., A_n ├ B, and proofs [T]: ├ A_1, [T]: ├ A_2, ..., [T]: ├ A_n, then there is a proof [T]: ├ B. In other words, if there is a proof [T]: A_1, A_2, ..., A_n ├ B, and A_1, A_2, ..., A_n are previously proved theorems of T, then B is a theorem of T as well.

Proof. Immediately, by Theorem 1.3.1.

Consistency

Sometimes, a seemingly plausible set of axioms allows deriving of contradictions (the most striking example − Russell's paradox in the "naive" set theory). A formula F is called a **contradiction** in the theory T, if [T]: ├ F and [T]: ├ ¬F, i.e., if T both proves and disproves F. Theories allowing to derive contradictions are called **inconsistent theories**. Thus, T is called a **consistent theory** if and only if T does not allow deriving of contradictions.

Normally, for a first order theory, the set of all theorems is infinite, and, therefore, **consistency cannot be verified empirically**. We may only hope to establish this desirable property by means of some **theoretical proof** (see Podnieks [1997], Section 5.4 for a more detailed discussion of this problem).

For theories adopting the above logical axioms, inconsistency is, in a sense, "the worst possible property". Indeed, the axiom L_{10}: $\neg B \to (B \to C)$ says that in an inconsistent theory anything is provable. In Exercise 1.4.2 we will − without L_{10} − prove 50% of it: [L_1, L_9, MP]: B, ¬B ├ ¬C. Thus, even without L_{10} (but with L_1): in an inconsistent theory anything is disprovable.

Is consistency enough for a theory to be "perfect", “non-empty” etc? In Section 4.3 we will prove the so-called Model Existence Theorem: if a first order theory is consistent, then there is a "model" (a kind of a "mathematical reality") where all its axioms and theorems are "true", i.e., a consistent theory is at least “non-empty”.

Completeness

If a formula contains free occurrences of variables, i.e., variables that are not bound by quantifiers (for example: $x=0 \vee x=1$), then the "truth value" of such formulas may depend on particular values assigned to the free variables. For example, the latter formula is "true" for x=1, yet it is "false" for x=2. Formulas that do not contain free occurrences of variables, are called **closed formulas**, for example:

$$\forall w \exists x (w < x \wedge prime(x)) \quad .$$

Closed formulas represent "definite assertions about the objects of a theory", and they are expected to be either "true", or "false". Or, in a first order **theory**, they are expected to be either provable, or disprovable (refutable). The above closed formula (stating that "there are infinitely many prime numbers") is provable − if our theory is first order arithmetic.

T is called a **complete theory** if and only if for each closed formula F in the language of T: [T]: ├ F or [T]: ├ ¬F, i.e., if and only if T proves or disproves any closed formula of its language. In other words: a complete theory can solve any problem from the domain of its competence.

In an **incomplete theory**, some closed formulas ("definite assertions about the objects of theory") can be neither proved, not disproved. Thus, an incomplete theory does not solve some of the problems from the domain of its competence.

Formally, according to this definition, an inconsistent theory is complete. Indeed, the axiom L_{10}: ¬B→(B→C) says that if a theory allows deriving a contradiction, then, in this theory, anything is provable, i.e., it is a complete theory.

Of course, if T would be both consistent and complete, then we could call it "absolutely perfect". Unfortunately, Gödel's Incompleteness Theorem says that all **fundamental mathematical theories are either inconsistent or incomplete**, i.e., none of them is absolutely perfect (see Mendelson [1997] or Podnieks [1997], Section 6.1).

Exercise 1.3.5 (optional). Re-formulate the above axiom system for a **many-sorted predicate language.**

1.4. The Flavor of Proving Directly

Theorem 1.4.1. [L_1, L_2, MP]: A→A for any formula A. What does it mean? It's the so-called **reflexivity property of implication**.

The following sequence of formulas represents a proof of the formula A→A (C can be here any formula, for, example, A itself):

(1)	(A→((C→A)→A))→((A→(C→A))→(A→A))	It's the axiom schema L_2: (B→(C→D))→((B→C)→(B→D)), with B = A, C = C→A, D = A.

(2) A→((C→A)→A)	It's the axiom schema L_1: B→(C→B), with B = A, C = C→A.
(3) (A→(C→A))→(A→A)	It follows from (1) and (2) by *Modus Ponens*.
(4) A→(C→A)	It's the axiom schema L_1: B→(C→B), with B = A, C = C.
(5) A→A	It follows from (3) and (4) by *Modus Ponens*.

As you can see, the proof is easy to verify as a correct one, but it seems hard to build it from scratch. "Why" should we take "the axiom L_2 with B = A, C = C→A, D = A" as the first step?

How could one invent a proof like the above one? One of the versions could be as follows. First, let's try to find an axiom, from which we could get A→A as a consequence. By trying L_1, i.e., B→(C→B), and setting B=C=A, we could obtain A→(A→A), a dead end. So, let's try L_2, i.e., (B→(C→D))→((B→C)→(B→D)). By setting B=D=A we obtain (A→(C→A))→((A→C)→(A→A)). It seems to be a good decision − because the first premise A→(C→A) is, in fact, L_1. Hence, by applying the MP rule, we obtain (A→C)→(A→A). Now, how to make A→C "provable"? Since C is, in fact, an arbitrary formula, we can replace C by C→A, obtaining (A→(C→A))→(A→A). The premise is here, again, L_1, hence, by applying the MP rule, we obtain A→A. Q.E.D. By performing all our replacements from the very beginning, we obtain the above proof of the formula A→A.

Hint. If, in your future formal proofs, you need to obtain an instance of the formula A→A, then just insert 5 formulas of the above proof (more precisely, insert the necessary instances of these formulas).

Theorem 1.4.2. [L_1, L_2, MP]: A→B, B→C ├ A→C, for any formulas A, B, C.

What does it mean? It's the so-called **Law of Syllogism** (by Aristotle), or the **transitivity property of implication**.

The following sequence of formulas represents a proof of the formula A→C from the hypotheses A→B and B→C:

(1) A→B	Hypothesis given.
(2) B→C	Hypothesis given.

(3)	(A→(B→C))→((A→B)→(A→C))	It's the axiom schema L_2: (B→(C→D))→((B→C)→(B→D)), with B = A, C = B, D = C.
(4)	(B→C)→(A→(B→C))	It's the axiom schema $\mathbf{L_1}$: B→(C→B), with B = B→C, C = A.
(5)	A→(B→C)	It follows from (2) and (4) by *Modus Ponens*.
(6)	(A→B)→(A→C)	It follows from (3) and (5) by *Modus Ponens*.
(7)	A→C	It follows from (1) and (6) by *Modus Ponens*.

Hint. If, in your future proofs, you need to apply the transitivity property of implication, then just insert the last 5 formulas of the above proof (more precisely, insert the corresponding instances of these formulas).

Note. Only the axiom schemas L_1 and L_2, and the inference rule *Modus Ponens* are used for proving the Theorems 1.4.1 and 1.4.2. Hence, **these theorems will remain valid for any logical system containing $\mathbf{L_1}$, $\mathbf{L_2}$ and *Modus Ponens*.**

Theorem 1.4.3. a) [L_1, L_2, MP]: A→(B→C) ├ B→(A→C). What does it mean? It's the so-called **Premise Permutation Law**.

b) [L_3, L_4, L_9, MP]: $\neg(A \wedge \neg A)$. What does it mean? It's the so-called **Law of Non-Contradiction**.

Proof. Do Exercises 1.4.1, 1.4.2.

Exercise 1.4.1. Build sequences of formulas representing the following proofs (only the axiom schemas L_1 and L_2 and *Modus Ponens* are necessary):

a) [L_1, MP]: A ├ B→A (a sequence of 3 formulas). What does it mean?

b) [L_2, MP]: A→B, A→(B→C) ├ A→C (a sequence of 5 formulas). What does it mean?

c) Prove Theorem 1.4.3(a) (a sequence of 9 formulas − thanks to Pavel Andreyev for the idea).

d) [L_1, L_2, MP]: A→(A→B) ├ A→B (easy solution – a sequence of 9 formulas, a smarter solution proposed by Arnold Ostrovsky – 8 formulas). What does it mean?

e) (for smart students) [L_1, L_2, MP]: (A→(A→B))→(A→B) (a sequence of 7 formulas! – a surprising sophisticated solution proposed by Ilmārs Cīrulis).

Exercise 1.4.2. Build sequences of formulas representing the following proofs:

b) [L_3-L_5, MP]: $A \wedge B \vdash B \wedge A$ (a sequence of 8 formulas). What does it mean?

c) [L_6-L_8, MP]: $A \vee B \to B \vee A$ (a sequence of 5 formulas). What does it mean?

d) [L_1, L_9, MP]: B, ¬B ⊢ ¬C (a sequence of 9 formulas). What does it mean? It's a weak form of the "crazy" axiom L_{10}: ¬A→(A→B). The axiom says: "Contradiction implies anything". As we see, in the minimal logic we can prove 50% of L_{10}: "Contradiction implies that all is wrong". Of course, this 50%-provability of L_{10} decreases the significance of the minimal logic accordingly.

e) Theorem 1.4.3(b) (a sequence of 5 formulas).

f) [L_1, L_8, L_{10}, MP]: $\neg A \vee B \to (A \to B)$ (a sequence of 5 formulas). What does it mean?

fx) [L_1, L_9, MP]: A→B, ¬B ⊢ ¬A (a sequence of 7 formulas). What does it mean? It's the so-called Modus Tollens rule.

g) [L_8, L_{11}, MP]: A→B, ¬A→B ⊢ B (a sequence of 7 formulas). What does it mean?

h) [L_1-L_8, MP]: $A \to B \vdash A \vee C \to B \vee C$ (a sequence of 11 formulas). What does it mean?

i) [L_1-L_{11}, MP]: $A \vee (A \to B)$ (a sequence of 14 formulas, a smarter solution by Ilmārs Cīrulis – 13 formulas). What does it mean? Does it mean anything at all?

Exercise 1.4.3 (optional, for smart students). Could you build shorter sequences proving the formulas of Exercise 1.4.1(c, d) and Exercise 1.4.2(b, d)? Evgeny Vihrov verified in 2011 that any proof of the formula of Exercise 1.4.1(d) will take more than 5 formulas.

1.5. Deduction Theorems

If, by assuming B as a hypothesis, we have proved C, then we have proved that B implies C. This natural way of reasoning is formalized in the so-called **deduction theorems** (introduced by Jacques Herbrand and Alfred Tarski):

J. Herbrand. Recherches sur la théorie de la démonstration. PhD Thesis, University of Paris, 1930 (approved in April 1929).

A. Tarski. Ueber einige fundamentale Begriffe der Metamathematik. "Comptes Rendus de Séances de la Société des Sciences et des Lettres de Varsovie, Classe III", 1930, Vol.23, pp. 22-29.

We will prove two such theorems – Deduction Theorem 1 (for propositional logic) and Deduction Theorem 2 (for predicate logic).

Theorem 1.5.1 (Deduction Theorem 1). If T is a first order theory, and there is a proof of

$$[T, MP]: A_1, A_2, ..., A_n, B \vdash C,$$

then there is a proof of

$$[L_1, L_2, T, MP]: A_1, A_2, ..., A_n \vdash B \rightarrow C.$$

In other words, having a *Modus Ponens* proof of C from the hypotheses A_1, A_2, ..., A_n, B, we can build a *Modus Ponens* proof of $B \rightarrow C$ from the hypotheses $A_1, A_2, ..., A_n$.

It appears that, usually, proving of [T, MP]: ... $B \vdash C$ is much easier (technically simpler) than proving of [T, MP]: ... $\vdash B \rightarrow C$.

Exercise 1.5.1 (optional, for smart students). Do not read the proof below. Try proving yourself.

Proof (thanks to Sergey Kozlovich for the idea, see also Kleene [1967], Exercise 10C). We must define a procedure allowing to convert any proof $[T, MP]: A_1, A_2, ..., A_n, B \vdash C$ into a proof $[L_1, L_2, T, MP]: A_1, A_2, ..., A_n \vdash B \rightarrow C$.

The easy way to do this would be using an induction by the number of formulas in the proof $[T, MP]: A_1, A_2, ..., A_n, B \vdash C$. But we will apply a more elegant idea. Any proof $[T, MP]: A_1, A_2, ..., A_n, B \vdash C$ is a sequence of formulas $F_1, F_2, \ldots, F_m$. We will replace each formula F_i by 3 or 5 formulas, the last of these being the formula $B \rightarrow F_i$, yet retaining our sequence as a valid

proof.

We must consider the following cases:

1) F is an axiom (i.e., an instance of a logical axiom or a non-logical axiom of T). Replace F by 3 formulas: F, F→(B→F), B→F. The second formula is an instance of L_1, the third formula is obtained from the first two ones by using *Modus Ponens*.

2) F is one of the hypotheses A_i. Replace F by 3 formulas: F, F→(B→F), B→F. The second formula is an instance of L_1, the third formula is obtained from the first two ones by using *Modus Ponens*.

3) F is B. Replace F by the 5 formulas from the proof of Theorem 1.4.1, where D can be here any formula, for, example, B itself:

(B→((D→B)→B))→((B→(D→B))→(B→B)) (an instance of L_2),

B→((D→B)→B) (an instance of L_1),

B→(D→B))→(B→B) (by *Modus Ponens*),

B→(D→B) (an instance of L_1),

B→B (by *Modus Ponens*).

The last formula is here, of course, B→F.

4) F is derived from some previous formulas F_i and F_j by *Modus Ponens*, F_i having the form F_j→F (i.e., F_j→F and F_j yield F by *Modus Ponens*). Then, the formulas B→F_j, B→(F_j→F) are already present in the converted proof (they appeared during the replacement operations applied to the formulas F_j and F_j→F). So, replace F by 3 formulas:

(B→(F_j→F))→((B→F_j)→(B→F)) (an instance of L_2),

(B→F_j)→(B→F) (by *Modus Ponens*),

B→F (by *Modus Ponens*).

Thus, what we have now, is a correct proof in [L_1, L_2, MP] that is using the hypotheses A_1, A_2, ..., A_n, but not B! The last formula of this proof is B→C (because C is the last formula of our initial proof [L_1, L_2, MP]: A_1, A_2, ..., A_n, B ├ C). Thus, we have a proof [L_1, L_2, MP]: A_1, A_2, ..., A_n ├ B→C.

Q.E.D.

The above proof of Deduction Theorem 1 includes, in fact, an **algorithm**

allowing to obtain a proof [L_1, L_2, MP]: A_1, A_2, ..., A_n ├ B→C from a given proof [L_1, L_2, MP]: A_1, A_2, ..., A_n, B ├ C. The resulting proof is much longer than the given one: if the given proof consists of *m* formulas, then the resulting proof consists of *3m* or *3(m–1)+5=3m+2* formulas.

Corollaries 1.5.1. 1) If there is a proof

$$[T, MP]: A_1, A_2, ..., A_n, B_1, B_2, ..., B_k \vdash C,$$

then there is a proof

$$[L_1, L_2, T, MP]: A_1, A_2, ..., A_n \vdash (B_1 \to (B_2 \to (... \to (B_k \to C)...))).$$

In particular, if [T, MP]: B ├ C, then [T, MP]: ├ B→C.

And, if [T, MP]: B, C ├ D, then [T, MP]: ├ B→(C→D).

2) If T includes (or proves) schemas L_1, L_2, then, if there is a proof [T, MP]: A_1, A_2, ..., A_n, B ├ C, then there is a proof [T, MP]: A_1, A_2, ..., A_n ├ B→C.

Proof. 1) By iterating Deduction Theorem 1.

2) If T is a theory which includes or proves the schemas L_1, L_2, then [L_1, L_2, T, MP] is equivalent to [T, MP]. Q.E.D.

Exercise 1.5.2 (optional, for smart students). In earlier versions of logical axioms, instead of the axiom L_2, in some texts, the following 3 axioms were used:

L_{21}: (A→(A→B))→(A→B),

L_{22}: (A→(B→C))→(B→(A→C)) (Premise Permutation Law),

L_{23}: (A→B)→((B→C)→(A→C)) (Law of Syllogism, or the transitivity property of implication).

Verify that both versions, i.e., [L_1, L_2, MP] and [L_1, L_{21}, L_{23}, L_{23}, MP] are equivalent. (Hints: a) See Section 2.1 to verify that [L_1, L_2, MP] proves L_{21}, L_{23}, L_{23}. b) Verify that [L_1, L_{21}, L_{23}, L_{23}, MP] proves L_2 either directly, or by proving the Deduction Theorem 1 for [L_1, L_{21}, L_{23}, L_{23}, MP].)

Exercise 1.5.3 (optional, thanks to Sergey Kozlovich for the idea).

a) Prove the following generalization of the *Modus Ponens* rule:

[L_1, L_2, MP]: $(D_1 \to (D_2 \to ... (D_k \to B)...))$, $(D_1 \to (D_2 \to ... (D_k \to (B \to C))...)$ ├ $(D_1 \to (D_2 \to ... (D_k \to C)...)$.

b) Prove the following generalization of the axiom L_{14} (formulas D_1, D_2, ..., D_k do not contain x as a free variable):

$[L_1, L_2, L_{14}, MP]: \vdash \forall x(D_1 \to (D_2 \to ...(D_k \to F(x))...) \to (D_1 \to (D_2 \to ... (D_k \to \forall x F(x))...)$.

Exercise 1.5.4 (optional, for smart students). Investigate the size (the number of formulas) of the proof of $[L_1, L_2, MP]: A_1, A_2, ..., A_n, \vdash B \to C$ as a function f(m) of the size m of the proof of $[L_1, L_2, MP]: A_1, A_2, ..., A_n, B \vdash C$. You may wish to report your result. We will publish your report on the web as an appendix to this book. The current record holder is Sergey Kozlovich, 2004: $f(m) \le 3m+2$. Improve this result, or prove that it is the best one possible.

Exercise 1.5.5 (optional, for smart students). Investigate the size (the number of instances of **atomic** formulas) of the proof of $[L_1, L_2, MP]: A_1, A_2, ..., A_n, \vdash B \to C$ as a function g(m) of the size m of the proof of $[L_1, L_2, MP]: A_1, A_2, ..., A_n, B \vdash C$. You may wish to report your result. We will publish your report on the web as an appendix to this book. The current record holder is Kirils Solovjovs, 2008: $g(m, n) \le 7m+24n-2$, where n is the number of instances of atomic formulas in the formula B. Improve this result, or prove that it is the best one possible.

Attention! *Generalization* involved...

Now, what, if in the proof of $A_1, A_2, ..., A_n, B \vdash C$ not only *Modus Ponens*, yet also *Generalization* rule is used?

We must be careful, because, trying to apply Deduction Theorem 1 formally, we can obtain crazy results. Indeed, having a formula F(x), by *Gen*, we obtain the formula $\forall x F(x)$. Thus, $F(x) \vdash \forall x F(x)$. If Deduction Theorem 1 could be extended to application of *Gen* without any restrictions, then we could conclude that $\vdash F(x) \to \forall x F(x)$. If this is true for any x, it is true also for x=2, hence, $\vdash F(2) \to \forall x F(x)$. Thus, if the number 2 is prime, then all numbers are prime?

So, let us try deriving a restricted formulation of the Deduction Theorem − it seems, we should **prohibit application of Gen to the variables that occur as free in B** − in the hypothesis "to be moved".

Theorem 1.5.2 (Deduction Theorem 2). If there is a proof

$$[T, MP, Gen]: A_1, A_2, ..., A_n, B \vdash C,$$

where, **after B appears in the proof**, *Generalization* **is not applied to the variables that occur as free in B**, then there is a proof of

$$[L_1, L_2, L_{14}, T, MP, Gen]: A_1, A_2, ..., A_n \vdash B \to C.$$

Proof. Let us modify the above proof of the Deduction Theorem 1.

We must define a procedure allowing to convert any correct proof $[T, MP, Gen]: A_1, A_2, ..., A_n, B \vdash C$ into a proof $[L_1, L_2, L_{14}, T, MP, Gen]: A_1, A_2, ...,$

$A_n \vdash B \to C$.

Unlike the proof of the Deduction Theorem 1, let us leave unchanged all the formulas of the proof of [T, MP]: $A_1, A_2, ..., A_n, B \vdash C$ **before B appears in the proof**. After this, **starting with B**, we will replace each formula F by 3 or 5 formulas, one of them being the formula $B \to F$.

So, F is B or appears after B. We must consider the following cases:

1), 2), 3) − like as in the proof of the Deduction Theorem 1.

4) F is derived from some previous formulas F_i and F_j by *Modus Ponens*, F_i having the form $F_j \to F$ (i.e., $F_j \to F$ and F_j yield F by *Modus Ponens*). Then, 4 sub-cases are possible.

4a) F_j and $F_j \to F$ both appear before B, i.e., they remain unchanged in the converted proof. Let us replace F by the following 3 formulas: F, $F \to (B \to F)$, $B \to F$. The second formula is an instance of L_1, the third formula is obtained by using *Modus Ponens* from the first two ones.

4b) F_j appears before B, and $F_j \to F$ is B or appears after B. Then, the formulas F_j and $B \to (F_j \to F)$ are already present in the converted proof. Let us replace F by the following 5 formulas:

$(B \to (F_j \to F)) \to ((B \to F_j) \to (B \to F))$ (an instance of L_2),

$(B \to F_j) \to (B \to F)$ (by *Modus Ponens*),

$Fj \to (B \to F_j)$ (an instance of L_1),

$B \to F_j$ (by *Modus Ponens*),

$B \to F$ (by *Modus Ponens*).

4c) F_j is B or appears after B, and $F_j \to F$ appears before B. Then, the formulas $B \to F_j$ and $F_j \to F$ are already present in the converted proof. Let us replace F by the following 5 formulas from the proof of Theorem 1.4.2:

$(B \to (F_j \to F)) \to ((B \to F_j) \to (B \to F))$ (an instance of L_2),

$(F_j \to F) \to (B \to (F_j \to F))$ (an instance of L_1),

$B \to (F_j \to F)$ (by *Modus Ponens*),

$(B \to F_j) \to (B \to F)$ (by *Modus Ponens*),

$B \to F$ (by *Modus Ponens*).

4d) F_j and $F_j \to F$ both are B or appear after B. Then, the formulas $B \to F_j$ and $B \to (F_j \to F)$ are already present in the converted proof (they appeared during the replacement operations applied to the formulas F_j and $F_j \to F$). Let us replace F by the following 3 formulas:

$(B \to (F_j \to F)) \to ((B \to F_j) \to (B \to F))$ (an instance of L_2),

$(B \to F_j) \to (B \to F)$ (by *Modus Ponens*),

$B \to F$ (by *Modus Ponens*).

5) F is derived from some previous formula F_i by *Generalization*, thus, F is in the form $\forall x F_i$. By the assumption of Deduction Theorem 2, **B does not contain free occurrences of x.** Then, 2 sub-cases are possible.

5a) F_i appears before B. Let us replace F by the following 3 formulas: F, $F \to (B \to F)$, $B \to F$. The second formula is an instance of L_1, the third formula is obtained by using *Modus Ponens* from the first two ones.

5b) F_i is B or appears after B. Then the formula $B \to F_i$ is already present in the converted proof (it appeared during the replacement operation applied to the formula F_i). Let us replace F by the following 3 formulas:

$\forall x(B \to F_i)$ (by *Generalization*, B does not contain x as a free variable),

$\forall x(B \to F_i) \to (B \to \forall x F_i)$ (B does not contain x as a free variable, thus, we have here an instance of L_{14}, and $\forall x F_i$ is F),

$B \to F$ (by *Modus Ponens*).

Thus, what we have now, is a correct proof in $[L_1, L_2, L_{14}, T, MP, Gen]$ that is using the hypotheses $A_1, A_2, ..., A_n$, but not B! The last formula of this proof is $B \to C$ (because C is the last formula our initial proof of $[T, MP, Gen]$: $A_1, A_2, ..., A_n, B \vdash C$). Thus, we have a proof $[L_1, L_2, L_{14}, T, MP, Gen]$: $A_1, A_2, ..., A_n \vdash B \to C$. Q.E.D.

In textbooks, the following restricted form of Deduction Theorem 2 is more popular:

Theorem 1.5.2A (Deduction Theorem 2A). If T is a first order theory, and there is a proof of

$$[T, MP, Gen]: A_1, A_2, ..., A_n, B \vdash C,$$

where *Generalization* **is not applied to the variables occurring as free in B**,

then there is a proof of

$$[L_1, L_2, L_{14}, T, MP, Gen]: A_1, A_2, ..., A_n \vdash B \to C.$$

Proof. Since, *Gen* is not applied to the variables occurring as free in B at all, it is not applied after B appears in the proof. Q.E.D.

Corollaries 1.5.2. 1) If there is a proof

$$[T, MP, Gen]: A_1, A_2, ..., A_n, B_1, B_2, ..., B_k \vdash C,$$

where *Generalization* is not applied to the variables appearing as free in the formulas $B_1, B_2, ..., B_k$, then there is a proof

$$[L_1, L_2, L_{14}, T, MP, Gen]: A_1, A_2, ..., A_n \vdash (B_1 \to (B_2 \to (... \to (B_k \to C)...))).$$

2) If B is a **closed** formula, and there is a proof

$$[T, MP, Gen]: A_1, A_2, ..., A_n, B \vdash C,$$

then there is a proof

$$[L_1, L_2, L_{14}, T, MP, Gen]: A_1, A_2, ..., A_n \vdash B \to C.$$

3) If T is a theory whose axioms include (or prove) the schemas L_1, L_2, L_{14}, then, if there is a proof

$$[T, MP, Gen]: A_1, A_2, ..., A_n, B \vdash C,$$

where *Generalization* is not applied to the variables occurring as free in B, then there is a proof of

$$[T, MP, Gen]: A_1, A_2, ..., A_n \vdash B \to C.$$

Proof. Similar to the proof of the above Corollaries of Deduction Theorem 1.

Exercise 1.5.6 (optional). In some other textbooks, a somewhat different system of logical axioms is used: instead of the axioms L_{14}, L_{15} and the *Generalization* rule the following two rules of inference are used:

$G \to F(x) \vdash G \to \forall x F(x)$ ($\forall$-Introduction);

$F(x) \to G \vdash \exists x F(x) \to G$ ($\exists$-Introduction).

Of course, here, G is a formula that does not contain x as a free variable. Verify that both systems are equivalent in all of their versions (minimal, constructive, and classical).

The surprising efficiency of Deduction Theorems

Try proving of

$$[L_1\text{- }L_4, MP]: \quad (A \to (B \to C)) \to (A \wedge B \to C) \quad . \qquad (*)$$

For proving directly – an almost impossible task!

But now, having Deduction Theorems, we can simplify the task of proving (*) and make it feasible. More precisely – we can make feasible the task **of proving that (*) is provable.** We will not propose a full proof of (*), we will propose a shorter proof that such a full proof exists!

Indeed, by introducing of two hypotheses, we can proceed as follows:

(1)	$A \to (B \to C)$	Hypothesis assumed.
(2)	$A \wedge B$	Hypothesis assumed.
(3)	$A \wedge B \to A$	Axiom L_3: $B \wedge C \to B$ with B = A, C = B.
(4)	$A \wedge B \to B$	Axiom L_4: $B \wedge C \to C$ with B = A, C = B.
(5)	A	By MP, from (2), (3).
(6)	B	By MP, from (2), (4).
(7)	B→C	By MP, from (1), (5).
(8)	C	By MP, from (6), (7).

Thus, we have established that

$$[L_3\text{- } L_4, \text{MP}]: \quad A \to (B \to C), A \wedge B \quad \vdash \quad C \quad .$$

Now, by Deduction Theorem 1,

$$[L_1\text{- } L_4, \text{MP}]: \quad A \to (B \to C) \quad \vdash \quad A \wedge B \to C \quad .$$

And let us apply this theorem once more,

$$[L_1\text{- } L_4, \text{MP}]: \vdash \quad (A \to (B \to C)) \to (A \wedge B \to C) \quad .$$

In fact, **we did not prove (*), i.e., we did not produce a sequence of formulas proving (*). We just proved that such a sequence does exist!** To produce it really, we must apply (twice!) the algorithm described in the proof of Deduction Theorem 1. As the result, we will obtain a full proof consisting of $3(3*8+2)+2=80$ formulas!

2. Propositional Logic

George Boole (1815-1864): "In 1854 he published *An Investigation into the Laws of Thought, on Which are founded the Mathematical Theories of Logic and Probabilities.* Boole approached logic in a new way reducing it to a simple algebra, incorporating logic into mathematics. He pointed out the analogy between algebraic symbols and those that represent logical forms. It began the algebra of logic called Boolean algebra which now finds application in computer construction, switching circuits etc." (according to MacTutor History of Mathematics archive).

See also:

G.Boole. The Calculus of Logic. *The Cambridge and Dublin Mathematical Journal*, vol. 3 (1848) (available online, published by **David R. Wilkins**).

2.1. Proving Formulas Containing Implication only

Experience once more the power of Deduction Theorems as means of "proving provability" of formulas.

Exercise 2.1.1. a) Produce a sequence of 4 formulas proving

$$[L_1, L_2, MP]: A\to(A\to B), A \vdash B.$$

b) Apply the algorithm described in the proof of Deduction Theorem 1 producing a sequence of $3*4+2=14$ formulas proving

$$[L_1, L_2, MP]: A\to(A\to B) \vdash A\to B.$$

c) (optional) Apply the algorithm once more – producing a sequence of 44 formulas proving

$$[L_1, L_2, MP]: (A\to(A\to B))\to(A\to B).$$

Compare these numbers with the currently best solution proposed by human solvers – a sophisticated sequence of 7 formulas instead of the generated 44 ones! See Exercise 1.4.1.

Attention! Be careful when assuming hypotheses.

For example, in order to prove the strange formula (the so-called Peirce's Law, see below) $((A\to B)\to A)\to A$, the only correct move would be assuming of $(A\to B)\to A$ as a hypothesis, and trying to prove A, i.e., trying to prove that $(A\to B)\to A \vdash A$.

Assuming of any further hypotheses would be wrong, for example, assuming

of A→B and A. Why? Because, after proving (easily) that A→B, A ├ A, by Deduction Theorem 1, it follows that A→B ├ A→A and ├ (A→B)→(A→A), or A ├ (A→B)→A and ├ A→((A→B)→A). Where do you see Peirce's Law here: ├ ((A→B)→A)→A?

Exercise 2.1.2. Use Deduction Theorem 1 to prove the following [L_1, L_2, MP]:

a) (**(A→B)→(A→C)**)→(**A**→(**B**→C)). Be careful when assuming hypotheses: assume (A→B)→(A→C), A, B – in this order, no other possibilities!

b) (**A→B**)→((**B→C**)→(**A**→C)). It's another version of the **Law of Syllogism** (by Aristotle), or the **transitivity property of implication**. Explain the difference between this formula and Theorem 1.4.2: A→B, B→C ├ A→C.

c) (**A→(B→C)**)→(**B**→(**A**→C)). It's another version of the **Premise Permutation Law.** Explain the difference between this formula and Theorem 1.4.3(a): A→(B→C) ├ B→(A→C).

2.2. Proving Formulas Containing Conjunction

The following simple theorem allows to make some or our "proofs of provability" shorter:

Theorem 2.2.1. a) (**C-introduction**) [L_5, MP]: A, B ├ $A \wedge B$.

b) (**C-elimination**) [L_3, L_4, MP]: $A \wedge B$ ├ A, $A \wedge B$ ├ B.

c) If there is a proof of [T, MP]: A_1, A_2, ..., A_n, A, B ├ C, then there is a proof of [T, L_3, L_4, MP]: A_1, A_2, ..., A_n, $A \wedge B$ ├ C.

Exercise 2.2.1. Prove (a, b) of Theorem 2.2.1.

Let us prove (c).

(1)	$A \wedge B$	Hypothesis given.
(2)	$A \wedge B \rightarrow A$	Axiom L_3: $B \wedge C \rightarrow B$ with B = A, C = B.
(3)	$A \wedge B \rightarrow B$	Axiom L_4: $B \wedge C \rightarrow C$ with B = A, C = B.
(4)	A	By MP, from (1) and (2).

(5) B By MP, from (1) and (3).

(6) $A_1, A_2, ..., A_n, A, B \vdash C$ Insert the given proof here.

Theorem 2.2.1(a, b) can be used as additional rules of inference allowing to further simplify our "proofs of provability", for example, the latter one:

(1) $A \wedge B$ Hypothesis given.

(4) A By C-elimination, from (1).

(5) B By C-elimination, from (1).

(6) $A_1, A_2, ..., A_n, A, B \vdash C$ Insert the given proof here.

Theorem 2.2.1 allows especially to simplify proving of **equivalences**. Let us remind that B↔C is defined as an abbreviation of $(B \to C) \wedge (C \to B)$. Of course, we will call B and C equivalent formulas if and only if $\vdash B \leftrightarrow C$. For example, by Theorem 1.4.1, $[L_1, L_2, MP]$: A→A, hence,

$[L_1, L_2, L_5, MP]$: $(A \to A) \wedge (A \to A)$, i.e.,

$[L_1, L_2, L_5, MP]$: A↔A.

Theorem 2.2.2. a) $[L_1, L_2, L_5, MP]$: (A→(B→C)) ↔ ((A→B)→(A→C)) (extension of the axiom L_2).

b) $[L_1\text{-}L_4, MP]$: $(A \to B) \wedge (B \to C) \to (A \to C)$ (another form of the **Law of Syllogism**, or **transitivity property of implication**).

Proof. a) Of course, (a) of the Exercise 2.1.2(b) is the reverse formula of the axiom L_2. Hence, by C-introduction we obtain (a).

Let us prove b):

(1) $(A \to B) \wedge (B \to C)$ Hypothesis assuned.

(2) A Hypothesis assumed.

(3) A→B By C-elimination, from (1).

(4) B→C By C-elimination, from (1).

(5) B By MP, from (2), (3).

(6) C By MP, from (4), (5).

Thus, we have established that $[L_1\text{-}L_4, MP]$: $(A \to B) \wedge (B \to C)$, $A \vdash C$. By

applying twice Deduction Theorem 1, $\vdash ((A \to B) \wedge (B \to C)) \to (A \to C)$.

Exercise 2.2.2. Prove the following [L_1- L_5, MP]:

a) A→B, A→C $\vdash A \to B \wedge C$. What does it mean?

b) $(A \to B) \wedge (A \to C) \to (A \to B \wedge C)$. What does it mean?

c) $A \to B \wedge C \vdash$ A→B. What does it mean?

d) $A \to B \wedge C \vdash$ A→C. What does it mean?

e) $(A \to B \wedge C) \to (A \to B) \wedge (A \to C)$. (Hint: apply transitivity of implication.) What does it mean? Hence,

[L_1- L_5, MP]: $(A \to B \wedge C) \leftrightarrow (A \to B) \wedge (A \to C)$.

Theorem 2.2.3 (properties of the conjunction connective). [L_1- L_5, MP]:

a) $A \wedge B \leftrightarrow B \wedge A$. What does it mean? **Conjunction is commutative**.

b) $A \wedge (B \wedge C) \leftrightarrow (A \wedge B) \wedge C$. What does it mean? **Conjunction is associative**.

c) $A \wedge A \leftrightarrow A$ **.** What does it mean? **Conjunction is idempotent.**

Exercise 2.2.3. Prove Theorem 2.2.3.

Exercise 2.2.4. Prove the following, [L_1- L_5, MP]:

a) $(A \to (B \to C)) \leftrightarrow (A \wedge B \to C)$. What does it mean?

b) $(A \to B) \to (A \wedge C \to B \wedge C)$. What does it mean? The converse formula $(A \wedge C \to B \wedge C) \to (A \to B)$ cannot be true. Explain, why.

c) A $\vdash B \leftrightarrow B \wedge A$. What does it mean?

Let us remind once more that the equivalence connective A↔B is defined as an abbreviation of $(A \to B) \wedge (B \to A)$.

Theorem 2.2.4 (properties of the equivalence connective). [L_1- L_5, MP]:

a) A↔A (**reflexivity**),

b) (A↔B)→(B↔A) (**symmetry**),

c) (A↔B)→((B↔C) →((A↔C)) (**transitivity**).

Exercise 2.2.5. Prove Theorem 2.2.4.

2.3. Proving Formulas Containing Disjunction

Like as Theorem 2.2.1, the following theorem allows to make some of our "proofs of provability" shorter.

Theorem 2.3.1. a) (**D-introduction**) [L_6, L_7, MP]: A $\vdash$ $A \vee B$; B $\vdash$ $A \vee B$

b) (**D-elimination**) If there is a proof

$$[T, MP]: A_1, A_2, ..., A_n, B \vdash D,$$

and a proof

$$[T, MP]: A_1, A_2, ..., A_n, C \vdash D,$$

then there is a proof

$$[T, L_1, L_2, L_8, MP]: A_1, A_2, ..., A_n, \quad B \vee C \quad \vdash D.$$

Exercise 2.3.1. a) Prove Theorem 2.3.1(a).

b) Elaborate on the following proof of Theorem 2.3.1(b). By Deduction Theorem 1, there are proofs of A_1, A_2, ..., $A_n \vdash B \to D$ and A_1, A_2, ..., $A_n \vdash C \to D$. Write down these two proofs one after another, append the axiom L_8 and $B \vee C$ as a hypothesis, and apply (three times) MP.

Theorem 2.3.1(b) is called "D-elimination" because of its mode of application: to prove $B \vee C \vdash D$, we "eliminate" the disjunction, by trying to prove separately $B \vdash D$ and $C \vdash D$, and after this, apply Theorem 2.3.1(b).

Theorem 2.3.2. a) [L_5, L_6-L_8, MP]: $A \vee B \leftrightarrow B \vee A$. What does it mean? **Disjunction is commutative**.

b) [L_1, L_2, L_5, L_6-L_8, MP]: $A \vee A \leftrightarrow A$. What does it mean? **Disjunction is idempotent**.

Exercise 2.3.2. Prove Theorem 2.3.2.

Theorem 2.3.3. Disjunction is associative:

$$[L_1, L_2, L_5, L_6\text{-}L_8, MP]: \quad A \vee (B \vee C) \leftrightarrow (A \vee B) \vee C \quad .$$

Proof. Let us prove, for example,

$$A \vee (B \vee C) \quad \vdash \quad (A \vee B) \vee C \quad .$$

By D-elimination, we can first try proving separately $A \vdash (A \vee B) \vee C$ and $B \vee C \vdash (A \vee B) \vee C$. So, part 1 of the proof:

(1) A Hypothesis given.

(2) $A \vee B$ By D-introduction, from (1).

(3) $(A \vee B) \vee C$ By D-introduction, from (2).

In part 2, we apply D-elimination once more, trying to prove separately $B \vdash (A \vee B) \vee C$ and $C \vdash (A \vee B) \vee C$:

Part 2a:

(1) B Hypothesis given.

(2) $A \vee B$ By D-introduction, from (1).

(3) $(A \vee B) \vee C$ By D-introduction, from (2).

Part 2b:

(1) C Hypothesis given.

(2) $(A \vee B) \vee C$ By D-introduction, from (1).

From Parts 2a, 2b, by D-elimination theorem, we obtain $B \vee C \vdash (A \vee B) \vee C$, and from Parts 1, 2 again, by D-elimination theorem, we obtain $A \vee (B \vee C) \vdash (A \vee B) \vee C$.

The second part of the proof (←) can be reduced in a similar way. Q.E.D.

Exercise 2.3.3. a) Prove

[L_1, L_2, L_6-L_8, MP]: $(A \to B) \to (A \vee C \to B \vee C)$. What does it mean? The converse formula $(A \vee C \to B \vee C) \to (A \to B)$ cannot be true. Explain, why.

b) Prove [L_1, L_2, L_6-L_8, MP]: A→B, C→D $\vdash A \vee C \to B \vee D$. What does it mean?

The following theorem resembles the well-known **distributive property** of (number) addition to multiplication: (a+b)c = ac+bc. Of course, the "dual" distributive property (of multiplication to addition) does not hold for numbers: ab+c=(a+c)(b+c) would imply ab+c=ab+ac+bc+cc, c=ac+bc+cc, and, if c<>0, then 1=a+b+c. Still, surprisingly, in logic,

Theorem 2.3.4. Conjunction is distributive to disjunction, and disjunction is distributive to conjunction:

a) [L_1-L_8, MP]: $(A \wedge B) \vee C \leftrightarrow (A \vee C) \wedge (B \vee C)$.

b) [L_1-L_8, MP]: $(A \vee B) \wedge C \leftrightarrow (A \wedge C) \vee (B \wedge C)$.

Exercise 2.3.4. Prove of Theorem 2.3.4. (Hint: apply D-elimination and D-introduction.)

2.4. Formulas Containing Negation – Minimal Logic

The following is another theorem allowing to make some of our "proofs of provability" shorter:

Theorem 2.4.1. (**N-elimination**) If there is a proof

$$[T, MP]: A_1, A_2, ..., A_n, B \vdash C,$$

and a proof

$$[T, MP]: A_1, A_2, ..., A_n, B \vdash \neg C,$$

then there is a proof

$$[T, L_1, L_2, L_9, MP]: A_1, A_2, ..., A_n \vdash \neg B.$$

What does it mean?

Proof. a) By Deduction Theorem 1, there are proofs of $A_1, A_2, ..., A_n \vdash B \to C$ and $A_1, A_2, ..., A_n \vdash B \to \neg C$. Write down these two proofs one after another, append the axiom L_9, and apply (twice) MP. Q.E.D.

Theorem 2.4.1 is called "N-elimination" because of its mode of application: to prove $\vdash \neg B$, we "eliminate" the negation, by trying to prove separately $B \vdash C$ and $B \vdash \neg C$, and after this, apply Theorem 2.4.1.

Theorem 2.4.2. a) $[L_1, L_2, L_9, MP]: A, \neg B \vdash \neg(A \to B)$. What does it mean?

b) $[L_1\text{-}L_4, L_9, MP]: A \wedge \neg B \to \neg(A \to B)$.

Proof.

(1)	A	Hypothesis given.
(2)	$\neg B$	Hypothesis given.
(3)	$A \to B$	Hypothesis (N-elimination).
(4)	B	By MP, from (1) and (3).

Thus, in (2) and (4) we have a contradiction. By N-elimination theorem, it follows that $A, \neg B \vdash \neg(A \to B)$. Q.E.D.

Exercise 2.4.1. a) Prove Theorem 2.4.2(b).

b) Prove that: $[L_1, L_2, L_9, MP]: (A \to \neg A) \to \neg A$. What does it mean? (Hint: apply N-elimination.)

Exercise 2.4.1A (optional, for smart students) Investigate the size (the number of formulas) of the proof of [L_1, L_2, L_9, MP]: A_1, A_2, ..., A_n ├ ¬B as a function f(k, m) of the sizes k, m of the proofs of [L_1, L_2, L_9, MP]: A_1, A_2, ..., A_n, B ├ C and A_1, A_2, ..., A_n, B ├ ¬C. You may wish to report your result. We will publish your report on the web as an appendix to this book. The current record holder is Aiga Romane, 2008: f(k, m) ≤ 3(k+m)+7. Improve this result, or prove that it is the best possible one.

Attention: non-constructive reasoning! In Section 2.6, we will use the classical logic [L_1-L_{11}, MP] to prove the converse of (b): $\neg(A \to B) \to A \wedge \neg B$, i.e., the equivalence $\neg(A \to B) \leftrightarrow A \wedge \neg B$. This converse formula cannot be proved in the constructive logic [L_1-L_{10}, MP] (see Section 2.8).

Theorem 2.4.3. [L_1, L_2, L_9, MP]: (A→B)→(¬B→¬A). What does it mean? It's the so-called **Contraposition Law**.

Note. The following rule form of Contraposition Law is called Modus Tollens:

$$[L_1, L_2, L_9, MP]: A \to B, \neg B \vdash \neg A, \text{ or, } \frac{A \to B\ ;\ \neg B}{\neg A} \ .$$

Attention: non-constructive reasoning! In Section 2.6, we will use the classical logic [L_1-L_{11}, MP] to prove the converse formula (¬B→¬A)→(A→B), i.e., the equivalence (A→B)↔(¬B→¬A). We will see also that this converse formula cannot be proved in the constructive logic [L_1-L_{10}, MP] (see Section 2.8).

Exercise 2.4.2. a) Prove Contraposition Law in two ways: by using N-elimination, and without it.

b) (optional, for smart students) Verify that, in our axiom system, Law of Non-Contradiction and Contraposition Law could be used instead of the axiom L_9. More precisely: prove L_9 in the logic [L_1-L_5, Law of Non-Contradiction, Contraposition Law, MP]. Be careful: do not use theorems depending on the axiom L_9.

Theorem 2.4.4. [L_1, L_2, L_9, MP]: A→¬¬A. What does it mean?

Attention: non-constructive reasoning! In Section 2.6, we will use the classical logic [L_1-L_{11}, MP] to prove the converse formula ¬¬A→A (the so-called **Double Negation Law**). We will see also (Section 2.8) that this converse formula cannot be proved in the constructive logic [L_1-L_{10}, MP].

Exercise 2.4.3. a) Prove Theorem 2.4.4.

b) Prove [L_1-L_9, MP]: (A→¬B)↔(B→¬A). What does it mean?

Attention: non-constructive reasoning! The formula (¬A→B)↔(¬B→A)

(very similar to the formula of Theorem 2.4.3) can be proved only in the classical logic. See Section 2.8.

Still, curiously, in the minimal logic we can prove:

Theorem 2.4.5. [L_1, L_2, L_9, MP]: ¬¬¬A↔¬A. What does it mean?

Proof. Indeed, by Theorem 2.4.4, ⊢ ¬A→¬¬¬A. By Contraposition Law, ⊢ (A→¬¬A)→(¬¬¬A→¬A). Hence, by Theorem 2.4.4 and MP, ⊢ ¬¬¬A→¬A. Q.E.D.

Theorem 2.4.5 (and some of the following formulas in this and in the next section containing double negations) may seem uninteresting to people believing unconditionally in the equivalence ¬¬A↔A. Still, it seems interesting (at least – for a mathematician) to obtain a general characterization of logical formulas that do not depend on the Law of Excluded Middle. In Section 2.7 we will use these formulas to prove the elegant and non-trivial *Glivenko's theorem*: a) A is provable in the classical propositional logic (in [L_1-L_{11}, MP]) if and only if ¬¬A is provable in the constructive propositional logic (in [L_1-L_{10}, MP]), b) ¬A is provable in the classical propositional logic if and only if ¬A is provable in the constructive propositional logic.

Theorem 2.4.6. a) [L_1, L_2, L_9, MP]: (¬A→A)→¬¬A. What does it mean?

b) [L_1, L_2, L_6, L_7, L_9, MP]: $\neg\neg(A\vee\neg A)$. What does it mean? This is a "weak form" of the Law of Excluded Middle that can be proved constructively. The formula $\neg\neg(A\vee\neg A)$ can be proved in the constructive logic, but $A\vee\neg A$ can't – as we will see in Section 2.8.

Exercise 2.4.4. Prove (a) and (b) of Theorem 2.4.6. The axiom L_{11} must not be used in these proofs! (Hint for (b): use axioms to derive a contradiction from $\neg(A\vee\neg A)$).)

We will need the results of the following theorem to prove Glivenko's Theorem in Section 2.7.

Theorem 2.4.7. [L_1-L_9, MP]:

a) (A→B)→(¬¬A→¬¬B). What does it mean?

b) ¬¬(A→B)→(¬¬A→¬¬B). What does it mean?

c) (A→(B→C))→(¬¬A→(¬¬B→¬¬C)). What does it mean?

d) ¬¬(A→B), ¬¬(B→C) ⊢ ¬¬(A→C). What does it mean?

e) ¬¬A, ¬¬(A→B) ⊢ ¬¬B. What does it mean?

The converse of (a): (¬¬A→¬¬B)→(A→B) cannot be proved in the constructive logic (see Section 2.8).

Proof. To prove (a), we must simply apply twice Contraposition Law: (A→B)→(¬B→¬A), and (¬B→¬A)→(¬¬A→¬¬B).

Now, let us prove (b).

(1)	¬¬(A→B)	Hypothesis assumed.
(2)	¬¬A	Hypothesis assumed.
(3)	¬¬A→((A→B)→¬¬B)	From (a), by transposing A→B and ¬¬A, by Premise Permutation Law.
(4)	(A→B)→¬¬B	From (2) and (3).
(5)	((A→B)→¬¬B)→(¬¬¬B→¬(A→B))	By Contraposition Law.
(6)	¬¬¬B→¬(A→B)	From (4) and (5).
(7)	(¬¬¬B→¬(A→B))→(¬¬(A→B)→¬¬¬¬B)	By Contraposition Law.
(8)	¬¬(A→B)→¬¬¬¬B	From (6) and (7).
(9)	¬¬¬¬B	From (1) and (8).
(10)	¬¬¬¬B→¬¬B	Theorem 2.4.5.
(11)	¬¬B	From (9) and (10).

Thus, by Deduction Theorem 1, ├ ¬¬(A→B)→(¬¬A→¬¬B).

Exercise 2.4.5. Prove (c) of Theorem 2.4.7. (Hint: apply (a) and (b).)

To prove d) ¬¬(A→B), ¬¬(B→C) ├ ¬¬(A→C), first, let us take (c) with A = A→B, B = B→C, C = A→C:

(1) ├ ((A→B)→((B→C)→(A→C)))→(¬¬(A→B)→(¬¬(B→C)→¬¬(A→C))).

(2)	├ (A→B)→((B→C)→(A→C)	By transitivity of implication and Deduction Theorem 1.
(3)	¬¬(A→B)	Hypothesis given.
(4)	¬¬(B→C)	Hypothesis given.
(5)	¬¬(A→C)	From (1), (3) and (4).

Of course, (e) is an easy consequence of (b). Q.E.D.

Theorem 2.4.8. [L_1-L_9, MP]:

a) $\neg\neg(A\wedge B)\leftrightarrow(\neg\neg A\wedge\neg\neg B)$. What does it mean?

b) $\neg\neg A\vee\neg\neg B\rightarrow\neg\neg(A\vee B)$. What does it mean?

Exercise 2.4.6. a) Prove Theorem 2.4.8(a). (Hint: apply Contraposition Law – several times, and Theorem 2.4.7(b).)

b) Prove Theorem 2.4.8(b). (Hint: apply D-elimination, D-introduction and Contraposition Law.)

Attention: non-constructive reasoning! The converse of (b): $\neg\neg(A\vee B)\rightarrow\neg\neg A\vee\neg\neg B$ cannot be proved in the constructive logic (see Section 2.8). If we simply succeed in deriving a contradiction from $\neg(A\vee B)$, then, perhaps, we do not have a method allowing to decide, which part of $\neg\neg A\vee\neg\neg B$ is true – ¬¬A, or ¬¬B?

Augustus de Morgan (1806-1871): "He recognised the purely symbolic nature of algebra and he was aware of the existence of algebras other than ordinary algebra. He introduced de Morgan's laws and his greatest contribution is as a reformer of mathematical logic." (according to MacTutor History of Mathematics archive).

Theorem 2.4.9.

a) [L_1, L_2, L_8, L_9, MP]: $\neg A\vee\neg B\rightarrow\neg(A\wedge B)$. It's the **constructive half of** the so-called **First de Morgan Law**. What does it mean?

b) [L_1-L_9, MP]: $\neg(A\vee B)\leftrightarrow\neg A\wedge\neg B$. It's the so-called **Second de Morgan Law**. What does it mean?

Attention: non-constructive reasoning! The second half of (a) – the converse implication $\neg(A\wedge B)\rightarrow\neg A\vee\neg B$ can be proved in the classical logic, but not in the constructive logic (see Section 2.8). Explain, why.

Proof. Let us prove (a).

(1)	$\neg A\vee\neg B$	Hypothesis assumed.
(2)	$A\wedge B$	Hypothesis (N-elimination).
(3)	A	C-elimination from (2).
(4)	B	C-elimination from (2).
(5)	$\neg A$ D-elimination from (1)	$\neg B$ D-elimination from (1)
(6)	$A;\neg A$ Contradiction.	$B;\neg B$ Contradiction.
(7)	$\neg A;\neg\neg A$ Unified contradiction.	$\neg A;\neg\neg A$ Unified contradiction.

The last step is necessary, if we wish to apply D-elimination theorem. In this

theorem, it is supposed that in both branches a common formula is derived, and not two different ones. To unify both contradictions, we applied here the "50% L_{10} theorem" ("contradiction implies that all is wrong") proved in Exercise 1.4.2: [L_1, L_9, MP]: $B;\neg B \vdash \neg A;\neg\neg A$. Since we are operating in the minimal logic, we could not apply L_{10} itself.

Exercise 2.4.7. Prove Theorem 2.4.9(b). (Hint: apply N- and D-elimination, the above-mentioned contradiction-unification included.)

Q.E.D.

Exercise 2.4.8. Prove:

a) [L_1-L_9, MP]: $(A\to B)\to\neg(A\wedge\neg B)$. What does it mean? Compare with Theorem 2.4.2(b): $A\wedge\neg B\to\neg(A\to B)$.

b) [L_1-L_8, MP]: $A\vee B\to((A\to B)\to B)$. What does it mean?

Attention: non-constructive reasoning! The converse implications of (a, b), $\neg(A\wedge\neg B)\to(A\to B)$ and $((A\to B)\to B)\to A\vee B$ cannot be proved in the constructive logic (see Section 2.8). Explain, why. Still, we will prove these formulas in the classical logic.

2.5. Formulas Containing Negation – Constructive Logic

In this book, **constructive logic** is used as a synonym of **intuitionistic logic**!

Constructive logic includes the axiom L_{10}: ¬B→(B→C), but rejects the Law of Excluded Middle L_{11}: $B\vee\neg B$ as a general logical principle.

Exercise 2.5.1. a) [L_{10}, MP]: A, ¬A ⊢ B. What does it mean?

b) [L_1, L_2, L_8, L_{10}, MP]: $A\vee B\to(\neg A\to B)$. What does it mean?

We will verify in Section 2.8 that the formula (b) cannot be proved in the minimal logic [L_1-L_9, MP] (and even not in [L_1-L_9, L_{11}, MP]), i.e., it cannot be proved without L_{10}.

Attention: non-constructive reasoning! The converse of (b), i.e., $(\neg A\to B)\to A\vee B$ cannot be proved in the constructive logic (see Section 2.8). Explain, why.

Theorem 2.5.1. a) [L_1, L_8, L_{10}, MP]: $\neg A\vee B\to(A\to B)$. What does it mean?

b) [L_1, L_2, L_6, MP]: $A \vee B \to (\neg A \to B) \vdash \neg A \to (A \to B)$. What does it mean? It means that the "natural" rule $A \vee B, \neg A \vdash B$ implies L_{10}!

We will verify in Section 2.8 that the formula Theorem 2.5.1(a) cannot be proved in the minimal logic [L_1-L_9, MP] (and even not in [L_1-L_9, L_{11}, MP]), i.e., it cannot be proved without L_{10}.

Attention: non-constructive reasoning! The converse of (a), i.e., $(A \to B) \to \neg A \vee B$ cannot be proved in constructive logic (see Section 2.8). Explain, why.

Proof. a) When proving (a), we cannot use Deduction Theorem 1 (because of the missing axiom L_2). So, let us simply build a sequence of 5 formulas representing the proof of (a):

(1)	$(\neg A \to (A \to B)) \to ((B \to (A \to B)) \to (\neg A \vee B \to (A \to B)))$	(Axiom L_8)
(2)	$\neg A \to (A \to B)$	Axiom L_{10}.
(3)	$B \to (A \to B)$	Axiom L_1.
(4)	$(B \to (A \to B)) \to (\neg A \vee B \to (A \to B))$	By MP, from (1), (2).
(5)	$\neg A \vee B \to (A \to B)$	By MP, from (4), (3).

b) Surprisingly, the rule $A \vee B, \neg A \vdash B$ seems to be a quite a "natural" logical principle, yet it cannot be proved without the axiom L_{10}! Why not? Because it implies L_{10}! Indeed,

(1)	$A \vee B \to (\neg A \to B)$	Hypothesis given.
(2)	¬A	Hypothesis assumed (¬A→(A→B) to prove!)
(3)	A	Hypothesis assumed.
(4)	$A \vee B$	By D-introduction, from (3).
(5)	B	By MP, from (1), (4) and (2).

Hence, by Deduction Theorem 1,

[L_1, L_2, L_6, MP]: $A \vee B \rightarrow (\neg A \rightarrow B)$ ├ ¬A→(A→B). Q.E.D.

Theorem 2.5.2. [L_1-L_{10}, MP]:

a) (¬¬A→¬¬B)→¬¬(A→B). It's the converse of Theorem 2.4.7(b). Hence, [L_1-L_{10}, MP]: ├ ¬¬(A→B)↔(¬¬A→¬¬B).

b) ¬¬A→(¬A→A). It's the converse of Theorem 2.4.6(a). Hence, [L_1-L_{10}, MP]: ¬¬A↔(¬A→A).

c) $A \vee \neg A \rightarrow (\neg \neg A \rightarrow A)$. What does it mean?

d) ¬¬(¬¬A→A). What does it mean? It's a "weak" form of the Double Negations Law – provable in constructive logic.

We will verify in Section 2.8 that formulas (a, b, c, d) cannot be proved in the minimal logic [L_1-L_9, MP] (and even not in [L_1-L_9, L_{11}, MP]), i.e., they cannot be proved without L_{10}.

Proof. Of course, (b) is an instance of the **axiom L_{10}.**

To prove (a), prove in Exercise 2.5.2 that ¬¬A→¬¬B, ¬(A→B) ├ ¬B, ¬¬B. Then, by N-elimination theorem, ├ (¬¬A→¬¬B)→¬¬(A→B).

To prove (c) and (d) do Exercise 2.5.2(b). Q.E.D.

Exercise 2.5.2. a) Prove that [L_1-L_{10}, MP]: ¬¬A→¬¬B, ¬(A→B) ├ ¬B, ¬¬B.

b) Prove (c) and (d) of Theorem 2.5.2.

Exercise 2.5.3. Prove that in [L_1-L_{10}, MP]:

a) A ├ $B \leftrightarrow B \vee \neg A$. What does it mean?

b) $B \vee (A \wedge \neg A) \leftrightarrow B$. What does it mean?

c) $((A \wedge \neg A) \wedge B) \vee C \leftrightarrow C$. What does it mean?

2.6. Formulas Containing Negation – Classical Logic

If you agree to adopt the axiom L_{11}: $B \vee \neg B$, i.e., the Law of Excluded Middle, you can prove, first of all:

Theorem 2.6.1. (Double Negation Law)

[L_1, L_2, L_8, L_{10}, L_{11}, MP]: ¬¬A → A.

Hence, [L_1-L_{11}, MP]: ¬¬A ↔ A.

Exercise 2.6.1. Prove Theorem 2.6.1.

We will verify in Section 2.8 that the formula ¬¬A → A cannot be proved in the minimal logic [L_1-L_9, MP] (and even not in [L_1-L_9, L_{11}, MP]), i.e., it cannot be proved without L_{10}.

In the minimal logic we proved Theorem 2.4.4: [L_1, L_2, L_9, MP]: A→¬¬A. Hence, [L_1-L_{11}, MP]: ¬¬A ↔ A.

Attention: non-constructive reasoning! The formula ¬¬A→A cannot be proved in the constructive logic, see Section 2.8. Why? Because it represents a kind of non-constructive reasoning. Indeed, imagine, you wish to prove that ∃xB(x). Assume the contrary, ¬ ∃ xB(x), and derive a contradiction. Thus you have proved the negation of ¬ ∃xB(x), i.e., ¬ ¬ ∃xB(x). To conclude ∃xB(x) from ¬ ¬ ∃xB(x), you need the Double Negation Law. Hence, by adopting this law as a logical principle, you would allow non-constructive existence proofs – if you prove ∃xB(x) by assuming ¬ ∃xB(x), and deriving a contradiction, then you may not obtain a method allowing to find a particular x satisfying B(x).

Theorem 2.6.2. [L_8, L_{11}, MP]: A→B, ¬A→B ⊢ B. Or, by Deduction Theorem 1, [L_1, L_2, L_8, L_{11}, MP]: (A→B)→((¬A→B)→B). What does it mean?

This formula cannot be proved in the constructive logic (see Section 2.8). Explain, why.

Exercise 2.6.2. Prove Theorem 2.6.2.

In the classical logic, you can prove also the converse of **Contraposition Law:**

Theorem 2.6.3. [L_1-L_{11}, MP]: (¬B→¬A)→(A→B). Hence, [L_1-L_{11}, MP]: (A→B) ↔ (¬B→¬A).

Attention: non-constructive reasoning! The formula (¬B→¬A)→(A→B) cannot be proved in the constructive logic, see Section 2.8. Explain, why.

Exercise 2.6.2. Prove that in [L_1-L_{11}, MP]:

a) Theorem 2.6.3. (Hint: apply Double Negation Law.)

b) (¬A→B)↔(¬B→A). Compare with Exercise 2.4.3(b): [L_1-L_9, MP]: (A→¬B)↔(B→¬A).

Attention: non-constructive reasoning! The formula (b) cannot be proved in the constructive logic, see Section 2.8.

Theorem 2.6.3. [L_1-L_9, L_{11}, MP]: ⊢ $\neg(A \wedge B) \rightarrow \neg A \vee \neg B$. Hence, [L_1-L_9,

L_{11}, MP]: $\vdash \neg(A \wedge B) \leftrightarrow \neg A \vee \neg B$.

The equivalence represents the **First de Morgan Law.**

Let us remind also the **Second de Morgan Law:**

$$[\mathbf{L_1\text{-}L_9}, \text{MP}]: \quad \neg(A \vee B) \leftrightarrow \neg A \wedge \neg B \quad .$$

Thus, both of de Morgan laws can be proved in [L_1-L_9, L_{11}, MP], i.e., they do not depend on the axiom L_{10}.

Proof. The constructive half of the First Law we proved in the minimal logic as Theorem 2.4.9(a): [L_1-L_9, MP]: $\neg A \vee \neg B \rightarrow \neg(A \wedge B)$. Let us prove the remaining half: [L_1-L_9, L_{11}, MP]: $\neg(A \wedge B) \rightarrow \neg A \vee \neg B$. The axiom L_{10} will not be used in this proof.

Let us apply a pretty **general method of proving formulas in the classical logic**: start the proof by introducing instances of L_{11} as hypotheses and applying D-elimination.

Our particular proof starts by assuming (0) $\neg(A \wedge B)$ as the hypothesis, after this we introduce $A \vee \neg A; B \vee \neg B$ as additional hypotheses. Then, by D-elimination, we obtain 4 branches starting with the following hypotheses:

1) $A; B$, 2) $A; \neg B$, 3) $\neg A; B$, 4) $\neg A; \neg B$.

In branches (2, 3, 4) we obtain the required formula $\neg A \vee \neg B$ simply by D-introduction.

In the branch (1):

(1)	A	Hypothesis assumed.
(2)	B	Hypothesis assumed.
(3)	$A \wedge B$	By C-introduction, from (1), (2). Contradiction with our initial hypothesis: (0) $\neg(A \wedge B)$.
(4)	$\neg A$	From (0) and (3), but not by L_{10}! By[L_1, L_9, MP]: B, ¬B ├ ¬C, proved in Exercise 1.4.2.
(5)	$\neg A \vee \neg B$	By D-introduction, from (4).

Thus, we have proved 4 cases:

$$[L_1\text{-}L_9, \text{MP}]: \quad \neg(A \wedge B); xA; yB \quad \vdash \quad \neg A \vee \neg B \quad ,$$

where each of *x, y* is either an empty symbol, or a negation symbol. By D-elimination theorem, we obtain 2 cases:

$$[L_1\text{-}L_9, MP]: \quad \neg(A \wedge B); xA; B \vee \neg B \quad \vdash \quad \neg A \vee \neg B \quad ,$$

and, finally: $[L_1\text{-}L_9, MP]: \quad \neg(A \wedge B); A \vee \neg B; B \vee \neg B \quad \vdash \quad \neg A \vee \neg B$ and

$[L_1\text{-}L_9, L_{11}, MP]: \quad \neg(A \wedge B) \quad \vdash \quad \neg A \vee \neg B$. Q.E.D.

It appears that, in the classical logic, we can **express implication by using negation and disjunction** (let us call it I-elimination). Indeed, we already know that $[L_1, L_8, L_{10}, MP]: \quad \neg A \vee B \rightarrow (A \rightarrow B)$ (Theorem 2.5.1).

Exercise 2.6.3. Prove that $[L_1\text{-}L_8, MP]: \quad A \vee C \quad \vdash \quad (A \rightarrow B) \rightarrow B \vee C$.

Hence, $[L_1\text{-}L_8, MP]: \quad A \vee \neg A \quad \vdash \quad (A \rightarrow B) \rightarrow \neg A \vee B$, i.e.,

Theorem 2.6.4. $[L_1\text{-}L_8, L_{11}, MP]: \quad (A \rightarrow B) \rightarrow \neg A \vee B$. Hence,

(**I-elimination**) $[L_1\text{-}L_{11}, MP]: \quad (A \rightarrow B) \leftrightarrow \neg A \vee B$.

Exercise 2.6.4. Prove that in $[L_1\text{-}L_{11}, MP]$:

a) $B \wedge (A \vee \neg A) \leftrightarrow B$. What does it mean?

b) $((A \vee \neg A) \vee B) \wedge C \leftrightarrow C$. What does it mean?

c) $((A \rightarrow B) \rightarrow B) \rightarrow A \vee B$. What does it mean? By adding the result of Exercise 2.4.7, $[L_1\text{-}L_{11}, MP]: \quad A \vee B \leftrightarrow ((A \rightarrow B) \rightarrow B)$ .Thus, in the classical logic, *disjunctions can be replaced by implications*.

Theorem 2.6.5. $[L_1\text{-}L_{11}, MP]: \quad \neg(A \rightarrow B) \rightarrow A \wedge \neg B$.

Hence, together with Theorem 2.4.2(b): $[L_1\text{-}L_{11}, MP]: \quad \neg(A \rightarrow B) \leftrightarrow A \wedge \neg B$.

Proof. Do Exercise 2.6.5(b).

Thus, in the classical logic, an implication is false if and only if the premise is true, and the conclusion is false.

In the constructive logic, if an implication is false, it follows only that the conclusion is false. Indeed, from L_1: $B \rightarrow (A \rightarrow B)$, by Contraposition Law: $[L_1, L_2, L_9, MP]: \quad \neg(A \rightarrow B) \rightarrow \neg B$. But the formula $\neg(A \rightarrow B) \rightarrow A$ can be proved in the classical logic only (see Theorem 2.8.5 in Section 2.8).

Exercise 2.6.5. Prove that in $[L_1\text{-}L_{11}, MP]$:

a) $(A \rightarrow B) \leftrightarrow \neg(A \wedge \neg B)$. What does it mean?

b) $\neg(A \rightarrow B) \leftrightarrow A \wedge \neg B$.

c) $A \vee B \leftrightarrow (\neg A \rightarrow B)$. What does it mean?

d) $A \wedge B \leftrightarrow \neg (A \rightarrow \neg B)$. What does it mean?

e) (optional, for smart students) Try detecting, which parts of these equivalences are provable in the constructive logic. (Hint: take a look at Theorem 2.8.5.)

Strange formulas

Exercise 2.6.6. Prove in the classical logic the following strange formulas. (Hint: the easy way – apply the method used above to prove Theorem 2.6.3.)

a) [L_1, L_2, L_6–L_8, L_{10}, L_{11}, MP]: $A \vee (A \rightarrow B)$. What does it mean? Does it mean anything at all? Compare with Exercise 1.4.2.

b) [L_1, L_2, L_6–L_8, L_{10}, L_{11}, MP]: $(A \rightarrow B) \vee (B \rightarrow A)$. What does it mean? Does it mean anything at all?

c) [L_1–L_{11}, MP]: $((A \rightarrow B) \rightarrow A) \rightarrow A$. What does it mean? Does it mean anything at all? It is the so-called **Peirce's Law** from:

C. S. Peirce. On the algebra of logic: A contribution to the philosophy of notation. *American Journal of Mathematics*, 1885, vol.7, pp.180-202.

2.7. Constructive Embedding: Glivenko's Theorem

Let us remind some of the results of previous sections concerning double negations:

Theorem 2.4.4. [L_1, L_2, L_9, MP]: A→¬¬A.

Theorem 2.4.5. [L_1–L_9, MP]: ¬¬¬A↔¬A.

Theorem 2.4.6(b). [L_1–L_9, MP]: $\neg \neg (A \vee \neg A)$. In this weak form, the Law of Excluded Middle can be proved constructively.

Theorem 2.4.7. [L_1–L_9, MP]: a) (A→B)→(¬¬A→¬¬B).

b) ¬¬(A→B)→(¬¬A→¬¬B).

c) (A→(B→C))→(¬¬A→(¬¬B→¬¬C)).

d) ¬¬(A→B), ¬¬(B→C) ⊢ ¬¬(A→C).

e) ¬¬A, ¬¬(A→B) ⊢ ¬¬B.

Theorem 2.4.8. [L_1-L_9, MP]:

a) $\neg \neg (A \wedge B) \leftrightarrow (\neg \neg A \wedge \neg \neg B)$.

b) $\neg\neg A \vee \neg\neg B \rightarrow \neg\neg(A \vee B)$.

Theorem 2.5.2. [L_1-L_{10}, MP]:

a) ($\neg\neg$A→$\neg\neg$B)→$\neg\neg$(A→B). It's the converse of Theorem 2.4.7(b).

d) $\neg\neg$($\neg\neg$A→A).

Theorem 2.6.1. [L_1-L_{11}, MP]: $\neg\neg$A ↔ A.

Does it mean that for **any** formula A:

If [L_1-L_{11}, MP]: ├ A, then [L_1-L_{10}, MP]: ├ $\neg\neg$A?

(The converse is obvious: if [L_1-L_{10}, MP]: ├ $\neg\neg$A, then [L_1-L_{11}, MP]: ├ A by Theorem 2.6.1.)

Imagine, we have a proof of [L_1-L_{11}, MP]: ├ A. It is a sequence of formulas R_1, R_2, ..., R_n, where R_n = A. If this sequence does not contain instances of the axiom L_{11}, then it is a proof of [L_1-L_{10}, MP]: ├ A as well. Hence, according to Theorem 2.4.4, [L_1-L_{10}, MP]: ├ $\neg\neg$A

If the sequence R_1, R_2, ..., R_n contains some instances of L_{11}, i.e., formulas having the form $B \vee \neg B$, then, according to Theorem 2.4.6(b)**,** we could try replacing each such formula by a sequence proving that [L_1-L_9, MP]: $\neg\neg(B \vee \neg B)$. It appears that **each** of the formulas $\neg\neg R_1$, $\neg\neg R_2$, ..., $\neg\neg R_n$ is provable in [L_1-L_{10}, MP].

a) If R_k is an instance of the axioms L_1-L_{10}, then [L_1-L_{10}, MP]: ├ $\neg\neg R_k$ (Theorem 2.4.4).

b) If R_k is an instance of the axiom L_{11}, then [L_1-L_{10}, MP]: ├ $\neg\neg R_k$ (Theorem 2.4.6(b)).

c) Now, let us assume that i, j < k, and R_i, R_j ├ R_k directly by MP, i.e., R_j is $R_i \rightarrow R_k$. We know already that [L_1-L_{10}, MP]: ├ $\neg\neg R_i$ and [L_1-L_{10}, MP]: ├ $\neg\neg(R_i \rightarrow R_k)$. By Theorem 2.4.7(b),

[L_1-L_9, MP]: ├ $\neg\neg(R_i \rightarrow R_k) \rightarrow (\neg\neg R_i \rightarrow \neg\neg R_k)$.

Hence, [L_1-L_{10}, MP]: ├ $\neg\neg R_k$.

Because A = R_n, we have proved the remarkable Glivenko's theorem from 1929:

V. Glivenko. Sur quelques points de la logique de M. Brouwer. *Academie Royale de Belgique, Bulletins de la classe des sciences*, 1929, ser.5, vol.15, pp.183-188.

Valery Ivanovich Glivenko (1897-1940) is best known by the so-called Glivenko-Cantelli theorem in probability theory.

Theorem 2.7.1 (Glivenko's Theorem).

$$[L_1\text{-}L_{11}, MP]: \vdash A \text{ if and only if } [L_1\text{-}L_{10}, MP]: \vdash \neg\neg A.$$

Or: a formula A is provable in the classical propositional logic if and only if its double negation $\neg\neg A$ is provable in the constructive propositional logic.

This theorem provides a kind of a "constructive embedding" for the classical propositional logic: any classically provable formula can be "proved" in the constructive logic, if you put two negations before it.

Corollary 2.7.1. $[L_1\text{-}L_{11}, MP]: \vdash \neg A$ if and only if $[L_1\text{-}L_{10}, MP]: \vdash \neg A$.

Or: a "negative" formula $\neg A$ is provable in the classical propositional logic if and only if it is provable in the constructive propositional logic.

Indeed, if $[L_1\text{-}L_{11}, MP]: \vdash \neg A$, then by Glivenko's theorem, $[L_1\text{-}L_{10}, MP]: \vdash \neg\neg\neg A$, and by Theorem 2.4.5, $[L_1\text{-}L_{10}, MP]: \vdash \neg A$. Q.E.D.

Exercise 2.7.1. Prove the following version of Glivenko's theorem (see Kleene [1952]):

a) If $[L_1\text{-}L_{11}, MP]: A_1, A_2, ..., A_n \vdash C$, then

$[L_1\text{-}L_{10}, MP]: \neg\neg A_1, \neg\neg A_2, ..., \neg\neg A_n \vdash \neg\neg C$.

b) If $[L_1\text{-}L_{11}, MP]: \neg A_1, \neg A_2, ..., \neg A_n, B_1, B_2, ..., B_p \vdash \neg C$, then

$[L_1\text{-}L_{10}, MP]: \neg A_1, \neg A_2, ..., \neg A_n , \neg\neg B_1, \neg\neg B_2, ..., \neg\neg B_p \vdash \neg C$.

2.8. Axiom Independence. Using Computers in Mathematical Proofs

If one of our axioms L_i could be proved by using the other axioms only, then we could simplify our logical system by dropping L_i as an axiom. A striking example:

Theorem 2.8.1. The axiom $\mathbf{L_9}$: $(A\to B)\to((A\to\neg B)\to\neg A)$ can be proved in $[L_1, L_2, L_8, L_{10}, L_{11}, MP]$.

This fact was established by Augusts Kurmītis (on the web, also: A. A. Kurmit):

A. A. Kurmitis. On independence of a certain axiom system of the propositional calculus. *Proc. Latvian State University*, 1958, Vol. 20, N3, pp. 21-25 (in Russian).

The following proof of L_9 in [L_1, L_2, L_8, L_{10}, L_{11}, MP] is due to Jānis Sedols.

First, let us establish that the formula (A→¬A)→¬A can be proved in [L_1, L_2, L_8, L_{10}, L_{11}, MP] (in Exercise 2.4.1 we established that [L_1, L_2, L_9, MP]: (A→¬A)→¬A):

(1)	$(A \to \neg A) \to ((\neg A \to \neg A) \to (A \vee \neg A) \to \neg A)$	Axiom L_8.
(2)	A→¬A	Hypothesis.
(3)	¬A→¬A	This is provable in [L_1, L_2, MP] (Theorem 1.4.1).
(4)	$A \vee \neg A$	Axiom L_{11}.
(4)	¬A	From (1), (2), (3) and (4), by MP.
(6)	(A→¬A)→¬A	By Deduction Theorem 1 (which is valid for any propositional system containing [L_1, L_2, MP]).

Now let us establish that in [L_1, L_2, L_{10}, MP]: A→B, A→¬B ├ A→¬A.

(7)	A→B	Hypothesis.
(8)	A→¬B	Hypothesis.
(9)	A	Hypothesis.
(9)	B	From (7), (9), by MP.
(10)	¬B	From (8), (9), by MP.
(11)	¬B→(B→¬A)	Axiom L_{10}.
(12)	¬A	From (9), (10) and (11) by MP.
(13)	A→B, A→¬B ├ A→¬A	By Deduction Theorem 1 (which is valid for any propositional system containing [L_1, L_2, MP]).

Finally, let us merge the proofs of (6) and (13), then by MP we obtain ¬A, i.e., [L_1, L_2, L_8, L_{10}, L_{11}, MP]: A→B, A→¬B ├ ¬A.

Now, by Deduction Theorem 1 (which is valid for any propositional system containing [L_1, L_2, MP]) we obtain the axiom L_9:

$$[L_1, L_2, L_8, L_{10}, L_{11}, MP]: (A \to B) \to ((A \to \neg B) \to \neg A).$$

Q.E.D.

What should we do after establishing that one of our axioms is "dependent"? Do you think, we should drop L_9 as an axiom of our logical system?

First, let's note that **we have proved L_9 by using three problematic axioms**: L_1, L_{10}, L_{11}. But L_9 itself is not problematic!

Secondly, L_9 **cannot** be proved in [L_1-L_8, L_{10}, MP] (see Theorem 2.8.2 below). Hence, if we would drop L_9, then, instead of a simple definition

$$\text{classical logic} = \text{constructive logic} + L_{11},$$

we would need a more complicated one:

$$\text{constructive logic} = \text{classical logic} - L_{11} + L_9.$$

So, let us retain L_9 in our list of logical axioms.

But now, the question of questions:

Is the Law of Excluded Middle an independent logical principle?

Are constructive logic and classical logic really different? Of course, they are, but how could one prove that?

Could we prove the Law of Excluded Middle (the axiom L_{11}: $B \vee \neg B$) by using the other axioms [L_1-L_{10}, MP] – as we proved L_9 in [L_1, L_2, L_8, L_{10}, L_{11}, MP]? If not, how could we demonstrate that this is impossible? How could we demonstrate that some logical principle is **independent**, i.e., that it cannot be derived from other principles?

Let us assume, we have designed an algorithm *q* that calculates for each formula A some its "property" q(A) in such a way that:

a) q(L_1) is true, q(L_2) is true, ..., q(L_{10}) is true (i.e., the axioms L_1-L_{10} possess the property *q*).

b) If q(A) is true and q(A→B) is true, then q(B) is true (i.e., *Modus Ponens* "preserves" the property q).

If so, then q(F) is true for all the formulas F that can be proved in [L_1-L_{10}, MP].

c) $q(L_{11})$ is false (i.e., L_{11} does not possess the property q).

If we could develop such an algorithm q, then this would demonstrate that L_{11} cannot be proved in [L_1-L_{10}, MP], i.e., that the *Law of Excluded Middle is an independent logical principle.*

Multi-valued logics

One of the ways way how to introduce remarkable properties of formulas are the so-called "multi-valued logics" or "many-valued logics", introduced by Jan Lukasiewicz and Emil Post:

J. Lukasiewicz. O logice trojwartosciowej. *Ruch Filozoficzny (Lwow)*, 1920, vol. 5, pp. 169-171

E. Post. Introduction to a general theory of elementary propositions. *Amer. journ. math.*, 1921, vol. 21, pp.163-195

Read more: Many-Valued Logic by Siegfried Gottwald in Stanford Encyclopedia of Philosophy.

For example, let us consider a kind of "three-valued logic", where 0 means "false", 1 – "unknown" (or NULL – in terms of SQL), and 2 means "true". Then it would be natural to define “truth values” of conjunction and disjunction as

$$A \wedge B = min(A, B) \ ;$$
$$A \vee B = max(A, B) \ .$$

But how should we define “truth values” of implication and negation?

A	B	$A \wedge B$	$A \vee B$	A→B
0	0	0	0	i_1
0	1	0	1	i_2
0	2	0	2	i_3
1	0	0	1	i_4
1	1	1	1	i_5
1	2	1	2	i_6
2	0	0	2	i_7
2	1	1	2	i_8

2	2	2	2	i_9

A	¬A
0	i_{10}
1	i_{11}
2	i_{12}

Thus, theoretically, we have here 3^9 = 19683 variants of implication definitions and 3^3 = 27 negation definitions.

Do you think, it would be natural to set the values of ¬A and A→B as follows?

A	¬A
0	2
1	1
2	0

A	B	A→B
0	0	2
0	1	2
0	2	2
1	0	1
1	1	1
1	2	2
2	0	0
2	1	1
2	2	2

Yes, it would be natural, if we would try building a natural three-valued logic, in which "1" would mean, indeed, "unknown". In this way we would obtain

the “natural” three-valued logic used, for example, for handling of NULL-values in SQL.

However, for our purpose, we must separate two different classes of formulas provable in the classical logic. For this task, the above “natural” logic is useless: under it, the axioms L_1- L_{11} “behave” irregularly (verify).

Thus, here, our aim must be just the opposite to inventing of a “natural” logic – creating of a “bad” logic.

Let us consider

"under our truth tables, formula A always takes "true" values (the value 2)"

as a kind of the above-mentioned "property" q(A). We are trying to prove that the axiom L_{11} cannot be derived from the axioms L_1- L_{10}. Hence, we must try to define our truth tables in such a way that:

a) the axioms L_1, L_2, ..., L_{10} always take the value 2,

b) *Modus Ponens* preserves taking always the valuec2 (i.e., if the formulas A and A→B are always 2, then B also is always 2),

c) the axiom L_{11} sometimes takes the value 0 or 1.

As we saw above, the truth tables, having these properties, cannot be 100% natural. So, we must explore the "unnatural" versions as well.

However, let us retain the “natural” definitions of conjunction and disjunction:

$$A \wedge B = min(A, B) \ ;$$
$$A \vee B = max(A, B) \ .$$

Exercise 2.8.1 (optional). Develop a simple (recursive) computer program receiving as input:

a) any truth tables of implication and negation,

b) Any formula F consisting of letters A, B, C, propositional connectives and parentheses,

and printing out "truth values" of the formula F, for example, if F = B→(A→B):

A	B	B→(A→B)
0	0	2
0	1	2
0	2	2
1	0	2
1	1	2

1	2	2
2	0	2
2	1	2
2	2	2

In this example the axiom L_1 always takes "true" values. Perhaps, we should be interested only in those variants of our truth tables that "satisfy" at least the axioms L_1, L_2, ..., L_8 forcing them always to take "true" values.

Thus, we consider

"under the selected truth tables, formula A always takes "true" values"

as a kind of the "property" q(A).

Will MP preserve this property? If A is "true", and A→B is "true", how could B be not? Let us consider the relevant part of the truth table for implication (the part where A is "true"):

A	B	A→B	
2	0	i_7	i_7=0,1
2	1	i_8	i_8=0,1
2	2	i_9	2

If we would consider only those variants of our truth tables where i_7 = 0 or 1, i_8 = 0 or 1, and i_9 = 2, then, if B would not be 2 for some values of its arguments, then A→B also would not be 2 for the same values of arguments.

Hence, if we restrict ourselves to truth tables with i_7 = 0 or 1, i_8 = 0 or 1, and i_9 = 2, then MP preserves the property of "being true". **i.e., from "true" formulas MP will derive only "true" formulas.**

The next idea: if we wish the axiom L_6: $A \to A \vee B$ always taking the value 2, then, **if A≤B, then A→B must be 2.**

Thus, of all the 3^9 = 19683 possible implication definition variants only the following 3*2*2 = 12 variants are worth of exploring:

A	B	A→B
0	0	2

0	1	2
0	2	2
1	0	i_4=0,1,2
1	1	2
1	2	2
2	0	i_7=0,1
2	1	i_8=0,1
2	2	2

Exercise 2.8.2. a) Verify that under any of these 12 implication definitions the axioms L_3, L_4, L_6, L_7 always take the value 2, i.e., you do not need testing these axioms any more.

b) For each of the axioms, L_1, L_2, L_5 and L_8, determine all the possible combinations of the values of i_4, i_7, i_8 forcing it to take always the value 2.

Note. The "intersection" of b) consists of 5 variants only.

Exercise 2.8.3 (optional) Extend your previous computer program by adding 6 nested loops: for i_4=0 to 2, for i_7=0 to 1, for i_8=0 to1, for i_aa=0 to 2, for i_b=0 to 2, for i_c=0 to 2. Let the program print out only those variants of truth tables that make "true" all the axioms L_1-L_8. (My own program produced 135 such variants, see the results file #00).

Thus, now we have 135 variants of truth tables which make "true" all the axioms L_1-L_8 and for which “truth” is retained when MP is applied. So, let us search among them for the variants that allow proving of axiom independence results we are interested in.

Axiom L_9

In Theorem 2.8.1 we established that the axiom **L_9**: (A→B)→((A→¬B)→¬A) can be proved in [L_1-L_8, L_{10}, L_{11}, MP]. Still,

Theorem 2.8.2. The axiom L_9 **cannot** be proved in [L_1-L_8, L_{10}, MP].

Proof. Let your program print out only those variants of truth tables that make "true" all the axioms L_1-L_8, and make: L_9 – not "true", and L_{10} – "true". My program yields 66 such variants, see the results file #01. Let us consider, for

example, the variant #33 (the unnatural clauses are marked bold):

Implication variant #3:
2 2 2 **2 2** 2 0 1 2 L1-L8 true.
Variant #33. Negation: 2 1 0 L9 not true. L10 true. L11 not true.

A	B	A→B
0	0	2
0	1	2
0	2	2
1	**0**	**2**
1	**1**	**2**
1	2	2
2	0	0
2	1	1
2	2	2

A	¬A
0	2
1	1
2	0

See the extended results file #1 for this variant.

Under this variant the axioms L_1-L_8 and L_{10} are "true". As we know, under this variant, by MP, from "true" formulas only "true" formulas can be derived. The axiom L_9 is not "true" under this variant:

A	B	(A→B)→((A→¬B)→¬A)
0	0	2
0	1	2
0	2	2

1	**0**	**1**
1	**1**	**1**
1	**2**	**1**
2	0	2
2	1	2
2	2	2

Hence, L_9 **cannot** be proved in [L_1-L_8, L_{10}, MP]. Q.E.D.

In a similar way, we can obtain some other independence results as well.

Axiom L_{10}

Theorem 2.8.3. The axiom $\mathbf{L_{10}}$: ¬B→(B→C) cannot be proved in the minimal logic [L_1-L_9, MP], and even not in [L_1-L_9, L_{11}, MP].

Proof. Let your program print out only those variants of truth tables that make "true" all the axioms L_1-L_8, and make: L_9 – "true", L_{10} – not "true", and L_{11} – "true". My program yields 6 such variants, see the results file #02. Let us consider, for example, the variant #1 (the unnatural clauses are marked bold):

Implication variant #1:
2 2 2 **0 2** 2 0 1 2 L1-L8 true.
Variant #1. Negation: 2 **2 1** L9 true. L10 not true. L11 true.

See the extended results file #2 for this variant.

Under this variant the axioms L_1-L_9 and L_{11} are "true". As we know, under this variant, by MP, from "true" formulas only "true" formulas can be derived. The axiom L_{10} is not "true" under this variant:

A	B	¬A→(A→B)
0	0	2
0	1	2
0	2	2
1	0	2
1	1	2

1	2	2
2	**0**	**0**
2	**1**	**1**
2	2	2

Hence, L_{10} **cannot** be proved in [L_1-L_9, L_{11}, MP]. Q.E.D.

Axiom L_{11}

Now, let us prove the main result of this section:

Theorem 2.8.4. The Law of Excluded Middle $\mathbf{L_{11}}$: $B \vee \neg B$ cannot be proved in the constructive propositional logic [L_1-L_{10}, MP]. **The Law of Excluded Middle is an independent logical principle**.

Proof. Let your program print out only those variants of truth tables that make "true" all the axioms L_1-L_8, and make: L_9 – "true", L_{10} – "true", L_{11} – not "true". My program yields only one such variant, see the results file #03:

Implication variant #1 (the unnatural clauses are marked bold):
2 2 2 **0 2** 2 0 1 2 L1-L8 true.
Variant #1. Negation: 2 **0 0** L9 true. L10 true. L11 not true.

See the extended results file #3 for this variant. (As we see, the implication definition coincides here with the one used above to "discredit" L_{10}, but the negation definition is different.)

Under this variant the axioms L_1-L_{10} are "true". As we know, under this variant, by MP, from "true" formulas only "true" formulas can be derived. The axiom L_{11} is not "true" under this variant:

B	¬B	$B \vee \neg B$
0	2	2
1	**0**	**1**
2	0	2

Hence, L_{11} **cannot** be proved in [L_1-L_{10}, MP]. Q.E.D.

The results file #03 proves also the following

Theorem 2.8.5 (thanks to Pavels Mihailovs for a correction**).** The following classically provable formulas cannot be proved in the constructive

propositional logic [L_1-L_{10}, MP]:

$$\neg\neg A \to A$$
$$(\neg B \to \neg A) \to (A \to B)$$
$$(\neg A \to B) \to (\neg B \to A)$$
$$(\neg\neg A \to \neg\neg B) \to (A \to B)$$
$$(A \to B) \to \neg A \vee B$$
$$((A \to B) \to B) \to A \vee B$$
$$((A \to B) \to A) \to A$$
$$\neg(A \wedge \neg B) \to (A \to B)$$
$$\neg(A \to B) \to A \wedge \neg B$$
$$A \vee (A \to B)$$

Indeed, all these formulas take non-"true" values under the truth tables from the proof of Theorem 2.8.4.

The following three formulas also cannot be proved in the constructive propositional logic, yet, unfortunately, the truth tables from our proof of Theorem 2.8.4 do not allow proving this:

$$\neg(A \wedge B) \to \neg A \vee \neg B$$
$$\neg\neg(A \vee B) \to \neg\neg A \vee \neg\neg B$$
$$(A \to B) \vee (B \to A)$$

Indeed, under the above truth tables, these formulas always take "true" values (see results file #03). However, this failure yields an interesting conclusion: **add these three formulas as additional axioms to [L_1-L_{10}, MP] – and L_{11} will remain still unprovable!**

Thus, we did not succeed in building a three-valued logic that would allow showing that the latter three formulas cannot be proved in the constructive propositional logic. Is it possible at all to build a multi-valued logic that would exactly separate constructively provable propositional formulas from the unprovable ones? Kurt Gödel showed in 1932 that this is impossible: **none of the finitely-valued logics "matches" exactly the constructive propositional logic!**

K. Gödel. Zum intuitionistischen Aussagenkalkül, *Anzeiger Akademie der Wissenschaften Wien*, Math.-naturwiss. Klasse, 1932, Vol. 69, pp.65-66.

Exercise 2.8.4 (optional, for smart students).

a)Verify somehow that the latter three formulas cannot be proved in the constructive propositional logic [L_1-L_{10}, MP]. Or, see Section 4.4 how to do this.

b) Verify that any of the following formulas could be used – instead of $B \vee \neg B$ – as the axiom L_{11} of the classical propositional logic: i) $(A \to B) \to \neg A \vee B$, ii) $\neg\neg B \to B$, iii) $\neg(A \to B) \to A$ (Hint: since all these formulas are provable in [L_1-L_{11}, MP], it remains to prove

L_{11} in [L_1-L_{10}, MP] + (i), in [L_1-L_{10}, MP] + (ii), and in [L_1-L_{10}, MP] + (iii)).

c) Verify that with ¬¬B→B instead of L_{11} the axiom L_{10} becomes 100% derivable from the other axioms. Perhaps, this is why many textbooks prefer the combination L_1-L_9 + ¬¬B→B as the axiom list for the classical propositional logic. But, then, we are forced to define the constructive propositional logic not as a subset of the classical one, but as the classical logic with the axiom ¬¬B→B **replaced** by the axiom L_{10}: ¬B→(B→C)!

Axiom L_{10} again...

Finally, let us check which of the main results of Section 2.5 (constructive logic) and Section 2.6 (classical logic) depend on the axiom L_{10}. Let your program print out only those variants of truth tables that make "true" all the axioms L_1-L_8, and make: L_9 – "true", L_{10} – not "true". My program yields 6 such variants, see the results file #04. Surprisingly, in all these variants L_{11} is "true" (thus, the results file #04 equals the results file #02). As the most productive appears

Implication variant #1:
2 2 2 0 2 2 0 1 2 L1-L8 true.
Variant #1. Negation: 2 2 1 L9 true. L10 not true. L11 true.

Constructively provable formulas:
Not true: $(A \vee B) \rightarrow ((\neg A) \rightarrow B)$
Not true: $((\neg A) \vee B) \rightarrow (A \rightarrow B)$
Not true: ((¬¬A)→(¬¬B))→(¬¬(A→B))
Not true: (¬¬A)→((¬A)→A)
Not true: $(A \vee (\neg A)) \rightarrow ((\neg \neg A) \rightarrow A)$
Not true: ¬¬((¬¬A)→A)

Classically provable formulas:
True: $(\neg \neg (A \vee B)) \rightarrow ((\neg \neg A) \vee (\neg \neg B))$
True: $(\neg (A \wedge B)) \rightarrow ((\neg A) \vee (\neg B))$
Not true: (¬¬A)→A
Not true: ((¬B)→(¬A))→(A→B)
Not true: ((¬A)→B)→((¬B)→A)
Not true: ((¬¬A)→(¬¬B))→(A→B)
True: $(A \rightarrow B) \rightarrow ((\neg A) \vee B)$
Not true: $((A \rightarrow B) \rightarrow B) \rightarrow (A \vee B)$
Not true: ((A→B)→A)→A
Not true: $(\neg (A \wedge (\neg B))) \rightarrow (A \rightarrow B)$
True: (A→B)→(((¬A)→B)→B)
Not true: $(\neg (A \rightarrow B)) \rightarrow (A \wedge (\neg B))$
Not true: $A \vee (A \rightarrow B)$

True: $(A \to B) \vee (B \to A)$
Not true: (A→B)→(((¬A)→(¬B))→(B→A))

Thus, the following constructively provable formulas cannot be proved in the minimal logic [L_1-L_9, MP] (and even not in [L_1-L_9, L_{11}, MP]), i.e., they cannot be proved without the axiom L_{10}:

$$(A \vee B) \to (\neg A \to B)$$
$$\neg A \vee B \to (A \to B)$$
$$(\neg\neg A \to \neg\neg B) \to \neg\neg(A \to B)$$
$$\neg\neg A \to (\neg A \to A)$$
$$A \vee \neg A \to (\neg\neg A \to A)$$
$$\neg\neg(\neg\neg A \to A)$$

And the following classically provable formulas cannot be proved without the axiom L_{10} **(**thanks to Pavels Mihailovs for a correction**)**:

$$\neg\neg A \to A$$
$$(\neg B \to \neg A) \to (A \to B)$$
$$(\neg A \to B) \to (\neg B \to A)$$
$$(\neg\neg A \to \neg\neg B) \to (A \to B)$$
$$((A \to B) \to B) \to A \vee B$$
$$((A \to B) \to A) \to A$$
$$\neg(A \wedge \neg B) \to (A \to B)$$
$$\neg(A \to B) \to A \wedge \neg B$$
$$A \vee (A \to B)$$
$$(A \to B) \to ((\neg A \to \neg B) \to (B \to A))$$

But how about the remaining five (classically provable) formulas (thanks to Stanislav Golubcov for the idea):

a) $(A \to B) \to \neg A \vee B$,
b) $\neg(A \wedge B) \to \neg A \vee \neg B$,
c) $\neg\neg(A \vee B) \to \neg\neg A \vee \neg\neg B$,
d) $(A \to B) \to ((\neg A \to B) \to B)$,
e) $(A \to B) \vee (B \to A)$?

Formulas (a, b, d) can be proved without without L_{10}, see Section 2.6.

Exercise 2.8.5. Show that also the formula (c) **can be** proved without L_{10}, i.e prove it in [L_1-L_9, L_{11}, MP]. Smart students: how about the remaining formula (e)?

Using computers in mathematical proofs

Do you trust the above "proofs of unprovability"? Of course, you do not need trusting my

(or your own) program generating the results files #00, #01, #02, #03 and #04. We used these files only to select the truth table variants allowing to prove our independence results. You may remove your worries by verifying directly ("manually") that under all the 3 truth table variants used above:

a) the axioms L_1-L_8 are true;

b) the axioms L_9, L_{10}, L_{11} and other formulas are true or not true according to the goal of each particular proof;

c) in all variants, $2\rightarrow0 \neq 2$, $2\rightarrow1 \neq 2$, $2\rightarrow2 = 2$, hence, from true formulas, Modus Ponens can derive only true formulas.

After this, you may forget about programs, the result does not depend any more on their correctness.

Unfortunately, in more complicated cases the situation does not allow for the above simple exit (imanual verification of the solution found by the computer). The historically first and most famous example is Four Colour Theorem (4CT, see in Wikipedia).

The proof of the Four Colour Theorem was completed in 1976 by Kenneth Appel (1932-2013) and Wolfgang Haken:

K. Appel and W. Haken, 'Every map is four colourable', Bulletin of the American Mathematical Society 82 (1976), 711–12.

K. Appel and W. Haken, 'Every map is four colourable, Part I: Discharging', Illinois Journal of Mathematics 21 (1977), 429–90.

K. Appel and W. Haken, 'Every map is four colourable, Part II: Reducibility', Illinois Journal of Mathematics 21 (1977), 491–567.

"The best-known, and most debated, instance is the use of computer analysis by Kenneth Appel and Wolfgang Haken of the University of Illinois in their 1976 proof of the four-colour conjecture (that four colours suffice to colour in any map drawn upon a plane in such a way that countries which share a border are given different colours). First put forward in 1852, the conjecture had become perhaps the most famous unsolved problem in mathematics, resisting a multitude of efforts at proof for over a century. Appel and Haken's demonstration rested upon computerized analysis, occupying 1,200 hours of computer time, of over 1,400 graphs. The analysis of even one of those graphs typically went beyond what an unaided human being could plausibly do: the ensemble of their demonstration certainly could not be checked in detail by human beings. In consequence, whether that demonstration constituted "proof" was deeply controversial..." (according to Donald MacKenzie. Computers and the Sociology of Mathematical Proof. In: *Trends in the History and Philosophy of Mathematics*, Odense: University of Southern Denmark, 2004, pp.67-86).

Technically, Appel and Haken created a set of 1476 small graphs ("configurations") and a set of more than 300 "discharging rules", such that (put somewhat roughly):

a) every of the 1476 configurations is "reducible" in the sense that if some planar graph G contains this configuration, then one can reduce G to a smaller graph G' such that any four-coloring of G' can be extended to a four-coloring of G;

b) any *minimal* non-four-colorable graph, "unavoidably", contains one of the 1476 configurations, which can be found by applying the "discharging rules".

Appel in 1998: "It is totally maddening that none of us seem to understand reducibility well enough to prove good general theorems about useful enough classes of reducible

configurations and thus computers must be used to show each individual configuration reducible." See Ken Appel on the 4CT proof, December 1998.

"… showing that a given configuration is reducible is fairly straightforward, but very laborious: the number of cases to consider increases geometrically to about 20,000,000" (Gonthier) for many of the configurations. Thus, Appel and Haken used a computer to verify and confirm that each of their 1476 configurations is reducible. In 1976, this verification process took about 1200 hours of computer time. This is why the correctness of the analysis, by far, "could not be checked in detail by human beings" (MacKenzie).

In 1995, Neil Robertson, Daniel P. Sanders, Paul Seymour and Robin Thomas proposed an "elegant … revision of the proof" (as put by Gonthier). They introduced new ideas, allowing to reduce the set of configurations to be checked to 633, and the set of discharging rules – to 32. However, even after this achievement, still, the proof remained non-human-verifiable: it "combined a textual argument, which could reasonably be checked by inspection, with computer code that could not [be checked by inspection]" (Gonthier, again). See The Four Color Theorem, November 13, 1995, by Robin Thomas (1962-2020), and the official publication:

N. Robertson, D. Sanders, P. Seymour, and R. Thomas. 'The Four-Colour Theorem', *Journal Combinatorial Theory*, Series B 70 (1997), 2–44.

In 2004, the above-mentioned 1995 proof was revised, improved and formalized by Georges Gonthier:

"… we have written a formal proof script that covers both the mathematical and computational parts of the proof. We have run this script through the Coq proof checking system..., which mechanically verified its correctness in all respects. Hence, even though the correctness of our proof still depends on the correct operation of several computer hardware and software components (the processor, its operating system, the Coq proof checker, and the Ocaml compiler that compiled it), none of these components are specific to the proof of the Four Colour Theorem."

"… the … 60,000 or so lines of the proof can be read for insight or even entertainment, but need not be reviewed for correctness. That is the job of the Coq proof assistant, a job for computers."

G. Gonthier. Formal proof–the four-color theorem. *Notices of the AMS* 55 (11), December 2008, 1382-1393

G. Gonthier. A computer-checked formalized proof of the Four Colour Theorem, 2016, pp.1-57

But it represented a correct (checked!) proof of 4CT from an accepted set of mathematical axioms.

Two other famous computer assisted mathematical proofs:

- In 1989, by using a Cray super-computer, Clement W. H. Lam finished his proof that finite projective plane of order 10 is impossible (for details see Projective plane in Wikipedia).

- In 1998, Thomas C. Hales finished his proof of Kepler conjecture about the densest arrangement of equal spheres in space (Johannes Kepler conjectured it in 1611, for details see Kepler conjecture in Wikipedia).

Visit The Coq Proof Assistant and Coq in Wikipedia.

3. Predicate Logic

3.1. Proving Formulas Containing Quantifiers and Implication only

Theorem 3.1.1. [L_1, L_2, L_{12}, L_{13}, MP]: $\forall x\, B(x) \to \exists x\, B(x)$. What does it mean? It prohibits "empty domains".

Proof. Indeed,

(1)	$\forall x\, B(x)$	Hypothesis assumed.
(2)	$\forall x\, B(x) \to B(x)$	Axiom L_{12}.
(3)	$B(x)$	By MP.
(4)	$B(x) \to \exists x\, B(x)$	Axiom L_{13}.
(5)	$\exists x\, B(x)$	By MP.

Thus, by [L_1, L_2, MP] Deduction Theorem 2, there is a proof of [L_1, L_2, L_{12}, L_{13}, MP]: $\forall x\, B(x) \to \exists x\, B(x)$. Q.E.D.

Theorem 3.1.2.

a) [L_1, L_2, L_{12}, L_{14}, MP, Gen]: $\forall x (B \to C) \to (\forall x\, B \to \forall x\, C)$. What does it mean?

b) [L_1, L_2, L_{12}-L_{15}, MP, Gen]: $\forall x (B \to C) \to (\exists x\, B \to \exists x\, C)$. What does it mean?

Proof. Let us prove(a).

(1)	$\forall x(B \to C)$	Hypothesis assumed.
(2)	$\forall xB$	Hypothesis assumed.
(3)	$\forall x(B \to C) \to (B \to C)$	Axiom L_{12}: $\forall xF(x) \to F(x)$.
(4)	$B \to C$	From (1) and (3), by MP.

(5) $\forall$xB→B	Axiom L_{12}: $\forall$xF(x)→F(x).
(6) B	From (2) and (5), by MP.
(7) C	From (4) and (6), by MP.
(8) $\forall$xC	From (7), by Gen.

In this proof, Gen is applied only to x, which is not a free variable in $\forall x(B \to C)$ and $\forall x B$. Thus, by Deduction Theorem 2, there is a proof of [L_1, L_2, L_{12}, L_{14}, MP, Gen]: $\forall x(B \to C) \to (\forall x B \to \forall x C)$.

Let us prove (b).

(1) $\forall$x(B→C)	Hypothesis assumed.
(2) $\forall$x(B→C)→(B→C)	Axiom L_{12}: $\forall$ xF(x)→F(x).
(3) B→C	From (1) and (2), by MP.
(4) C→$\exists$xC	Axiom L_{13}: F(x)→ $\exists$xF(x).
(5) B→$\exists$xC	From (3) and (4), by transitivity of implication [L_1, L_2, MP].
(6) $\forall$x(B→$\exists$xC)	From (5), by Gen.
(7) $\forall$x(B→$\exists$xC)→($\exists$xB→$\exists$xC)	Axiom L_{15}: $\forall$x(F(x)→G)→($\exists$xF(x)→G) (no free occurrences of x in $\exists$xC).
(8) $\exists$xB→ $\exists$ xC	From (6) and (7), by MP.

In this proof, *Gen* is applied only to x, which is not a free variable in $\forall x(B \to C)$. Thus, by [L_1, L_2, L_{14}, MP, Gen] Deduction Theorem 2, there is a proof of [L_1, L_2, L_{12}-L_{15}, MP, Gen]: $\forall x(B \to C) \to (\exists x B \to \exists x C)$.

Q.E.D.

Now, let us prove two theorems allowing to make our proofs shorter.

Theorems 3.1.3. If F is any formula, then:

a) (**U-introduction**) [Gen]: $F(x) \vdash \forall x F(x)$.

b) (**U-elimination**) [L_{12}, MP, Gen]: $\forall x F(x) \vdash F(x)$. What does it mean?

c) (**E-introduction**) [L_{13}, MP, Gen]: $F(x) \vdash \exists x F(x)$. What does it mean?

Proof. Obvious.

Theorems 3.1.4. If F is any formula, and G is a formula that does not contain free occurrences of x, then:

a) (**U2-introduction**) [L_{14}, MP, Gen] $G \to F(x) \vdash G \to \forall x F(x)$. What does it mean?

b) (**E2-introduction**) [L_{15}, MP, Gen]: $F(x) \to G \vdash \exists x F(x) \to G$. What does it mean?

Proof. Let us prove (a). The following sequence of formulas represents a proof of the formula $G \to \forall x F(x)$ from the hypothesis $G \to F(x)$:

(1) G→F(x)	Hypothesis given.
(2) ∀x(G→F(x))	Follows from (1) by *Gen*.
(3) ∀x(G→F(x))→(G→∀xF(x))	The axiom schema $\mathbf{L_{14}}$ (no free occurrences of x in G).
(4) G→∀xF(x)	From (2) and (3) by MP.

The proof of (b) is similar. Q.E.D.

Attention! Note that U-introduction, U2-introduction and E2-introduction involve application of *Gen*, so, these rules fall under the restriction of Deduction Theorem 2.

Let us apply the new rules to simplify the proof of Theorem 3.1.2.

(a):

(1) ∀x(B→C)	Hypothesis assumed.
(2) ∀xB	Hypothesis assumed.
(3) B→C	From (1), by U-elimination.
(4) B	From (2), by U-elimination.
(5) C	From (3) and (4), by MP.
(6) ∀xC	From (5), by Gen.

(b):

(1)	$\forall x(B \rightarrow C)$	Hypothesis assumed.
(2)	$B \rightarrow C$	From (1), by U-elimination.
(3)	$C \rightarrow \exists xC$	Axiom L_{13}: $F(x) \rightarrow \exists xF(x)$.
(4)	$B \rightarrow \exists xC$	From (2) and (3), by transitivity of implication [L_1, L_2, MP].
(5)	$\exists xB \rightarrow \exists xC$	From (4), by E2-introduction.

Theorem 3.1.5.

a) [L_1, L_2, L_5, L_{12}, L_{14}, MP, Gen]: $\forall x \forall yB(x, y) \leftrightarrow \forall y \forall xB(x, y)$. What does it mean?

b) [L_1, L_2, L_5, L_{13}, L_{15}, MP, Gen]: $\exists x \exists yB(x, y) \leftrightarrow \exists y \exists xB(x, y)$. What does it mean?

c) [L_1, L_2, L_{12}-L_{15}, MP, Gen]: $\exists x \forall yB(x, y) \rightarrow \forall y \exists xB(x, y)$. What does it mean? The converse implication $\forall x \exists yB(x, y) \rightarrow \exists y \forall xB(x, y)$ cannot be true. Explain, why.

Proof.

Exercise 3.1.1. Prove (a) and (c) of Theorem 3.1.5.

Let us prove (b).

(1)	$B(x, y) \rightarrow \exists xB(x, y)$	Axiom L_{13} with $F(x) = B(x, y)$.
(2)	$\exists xB(x, y) \rightarrow \exists y \exists\, xB(x, y)$	Axiom L_{13} with $F(y) = \exists xB(x, y)$.
(3)	$B(x, y) \rightarrow \exists y \exists xB(x, y)$	From (1) and (2), by transitivity of implication [L_1, L_2, MP].
(4)	$\exists yB(x, y) \rightarrow \exists y \exists\, xB(x, y)$	From (3), by E2-introduction.
(5)	$\exists x \exists yB(x, y) \rightarrow \exists y \exists xB(x, y)$	From (4), by E2-introduction.

The proof of the converse implication [L_1, L_2, L_{13}, L_{15}, MP, Gen]: $\exists y \exists x\, B(x, y) \rightarrow \exists x \exists yB(x, y)$ is identical.

By C-introduction [L_5, MP] we obtain the equivalence (b). Q.E.D.

Exercise 3.1.2. Prove in the constructive logic,

[L_1-L_{10}, L_{12}-L_{15}, MP, Gen]: $\exists x(B(x) \rightarrow C(x)) \rightarrow (\forall xB(x) \rightarrow \exists xC(x))$.

Dropping quantifiers

Theorem 3.1.6. If the formula B does not contain free occurrences of x, then [L_1-L_2, L_{12}-L_{15}, MP, Gen]: $(\forall x\, B)\leftrightarrow B\,; (\exists x\, B)\leftrightarrow B$, i.e., quantifiers $\forall x\,; \exists x$ can be dropped or introduced as needed.

Proof. By L_{12} and L_{13}: $\forall x\, B\rightarrow B\,; B\rightarrow \exists x\, B$. By L_{14} and L_{15}:

$\forall x(B\rightarrow B)\rightarrow(B\rightarrow \forall x\, B)$; $\forall x(B\rightarrow B)\rightarrow(\exists x\, B\rightarrow B)$. By L_1 and L_2, MP and Gen: $\forall x(B\rightarrow B)$. Q.E.D.

3.2. Formulas Containing Negations and a Single Quantifier

Attention: non-constructive reasoning! ¬∀xB→∃x¬B. This formula is accepted in the classical logic: if not all x-s possess the property B, then there is an x that does not possess B. It represents non-constructive reasoning in its ultimate form: let us assume, all x-s possess the property B, if we succeed in deriving a contradiction from this assumption, then – what? Is this a proof that there is a particular x that does not possess the property B? Does our proof contain a method allowing to build at least one such x? If not, do we have a "real" proof of ∃x¬B?

How many formulas can be built of the formula B by using negations and a single quantifier?

¬¬¬¬¬¬¬¬¬¬∀x¬¬¬¬¬¬¬¬¬¬B

¬¬¬¬¬¬¬¬¬¬∃x¬¬¬¬¬¬¬¬¬¬B

Classical logic

In the classical logic: [L_1-L_{11}, MP]: ⊢ ¬¬A↔A, hence, any number of negations can be reduced to zero or one, and we have to investigate the following 8 formulas, in fact, 4 pairs or equivalent formulas:

¬∀x¬B↔∃xB; ¬∀xB↔∃x¬B; ∀x¬B↔¬∃xB; ∀xB↔¬∃x¬B;

All of them are provable in the classical logic. Indeed, the second equivalence can be obtained from the first one by replacing B by ¬B, the third one – by Contraposition Law. And finally, the fourth equivalence can be obtained from the third one by replacing B by ¬B. Thus, we need to prove only the first equivalence:

Theorem 3.2.1. In the classical logic,

$$[L_1\text{-}L_{15}, MP, Gen]: \neg\forall x\neg B \leftrightarrow \exists xB.$$

Proof. a) $\vdash \neg\forall x\neg B\rightarrow\exists xB$ (this part can be proved in the classical logic only).

(1)	$B\rightarrow\exists xB$	Axiom L_{13}: $F(x)\rightarrow\exists xF(x)$.
(2)	$\neg\exists xB\rightarrow\neg B$	From (1), by Contraposition Law.
(3)	$\neg\exists xB\rightarrow\forall x\neg B$	From (2), by U2-introduction [L_{14}, Gen]
(4)	$\neg\forall x\neg B\rightarrow\neg\neg\exists xB$	From (3), by Contraposition Law.
(5)	$\neg\forall x\neg B\rightarrow\exists xB$	From (4), in **Classical logic**, $\vdash \neg\neg B \rightarrow B$, and transitivity of implication.

b) $\vdash \exists x\, B\rightarrow\neg\forall x\neg B$ (this part can be proved in the constructive logic).

(1)	$\forall x\neg B\rightarrow\neg B$	Axiom L_{12}: $\forall xF(x)\rightarrow F(x)$.
(2)	$\neg\neg B\rightarrow\neg\forall x\neg B$	From (1), by Contraposition Law.
(3)	$B\rightarrow\neg\forall x\neg B$	From (4), in **Constructive logic**, $\vdash B \rightarrow \neg\neg B$, and transitivity of implication.
(4)	$\exists xB\rightarrow\neg\forall x\neg B$	From (3), by E2-introduction [L_{15}, Gen].

Q.E.D.

Constructive logic

Here we have a weaker Theorem 2.4.5 [L_1-L_9, MP]: $\neg\neg\neg A\leftrightarrow\neg A$. Hence, any number of negations can be reduced to zero, one, or two, and thus we obtain 3*2*3 = 18 formulas to be investigated. The following Table 3.2 represents the results of this investigation from

A.Heyting. On weakened quantification. *Journal of Symbolic Logic*, 1936, vol.11, pp.119-121 (see also Kleene [1952], Section 3.5).

Legend of Table 3.2. a) In the classical logic, within each of the 4 groups all formulas are equivalent, for example, in group III: $\neg\forall xB\leftrightarrow\exists x\neg B$. Of course, formulas belonging to different groups cannot be equivalent (explain, why).

b) Two formulas within a group are constructively equivalent if and only if they have no separating lines between them. For example, in group II: constructively, $\neg\forall x\neg B\leftrightarrow\neg\neg\exists xB$, but not $\neg\forall x\neg B\leftrightarrow\exists xB$ (explain, why). All

the formulas of the group IV are constructively equivalent.

Table 3.2

I $\forall xB$ --- $\neg\neg\forall xB$ === $\forall x\neg\neg B$ $\neg\neg\forall x\neg\neg B$ $\neg\exists x\neg B$	**III** $\exists x\neg B$ --- $\neg\neg\exists x\neg B$ $\neg\forall x\neg\neg B$ === $\neg\forall xB$
II $\exists xB$ --- $\exists x\neg\neg B$ --- $\neg\neg\exists xB$ $\neg\neg\exists x\neg\neg B$ $\neg\forall x\neg B$	**IV** $\forall x\neg B$ $\neg\neg\forall x\neg B$ $\neg\exists x\neg\neg B$ $\neg\exists xB$

c) If two formulas F_1, F_2 within a group (F_1 – above, F_2 – below) are separated by a **single line**, then: constructively, $F_1\rightarrow F_2$, and $\neg\neg(F_2\rightarrow F_1)$, but not $F_2\rightarrow F_1$. For example, in group II: constructively, $\exists xB\rightarrow\neg\forall x\neg B$, and $\neg\neg(\neg\forall x\neg B\rightarrow\exists xB)$, but not $\neg\forall x\neg B\rightarrow\exists xB$ (explain, why).

d) If two formulas F_1, F_2 within a group (F_1 – above, F_2 – below) are separated by a **double line**, then: constructively, $F_1\rightarrow F_2$, but not $F_2\rightarrow F_1$, and even not $\neg\neg(F_2\rightarrow F_1)$. For example, in group III: constructively, $\exists x\neg B\rightarrow\neg\forall xB$, but not $\neg\forall xB\rightarrow\exists x\neg B$, and even not $\neg\neg(\neg\forall xB\rightarrow\exists x\neg B)$ (try explaining, why). Thus, the implication $\neg\forall xB\rightarrow\exists x\neg B$ could be called **“super-non-constructive”**.

End of **Legend**.

Let us prove the implications necessary for the positive part of the above legend to be true.

Note. Proofs necessary for the negative part are not considered in the current

version of this book. The necessary methods are considered in Section 4.4.

Group I

I-1. Constructively, [L_1, L_2, L_9, MP]: $\forall xB \to \neg\neg\forall xB$.

Immediately, by [L_1, L_2, L_9, MP]: $A \to \neg\neg A$.

I-2. Constructively, [L_1-L_9, L_{12}, L_{14}, MP, Gen]: $\neg\neg\forall xB \to \forall x\neg\neg B$.

(1)	$\forall xB \to B$	Axiom L_{12}: $\forall xF(x) \to F(x)$
(2)	$\neg\neg\forall xB \to \neg\neg B$	Theorem 2.4.7(a) [L_1-L_9, MP]: $(A \to B) \to (\neg\neg A \to \neg\neg B)$
(3)	$\neg\neg\forall xB \to \forall x\neg\neg B$	U2-introduction [L_{14}, Gen]

I-3. Constructively, [L_1, L_2, L_9, MP]: $\forall x\neg\neg B \to \neg\neg\forall x\neg\neg B$.

Immediately, by [L_1, L_2, L_9, MP]: $A \to \neg\neg A$.

I-4. Constructively, [L_1, L_2, L_9, L_{12}, L_{15}, MP, Gen]: $\neg\neg\forall x\neg\neg B \to \neg\exists x\neg B$.

(1)	$\forall x\neg\neg B \to \neg\neg B$	Axiom L_{12}: $\forall xF(x) \to F(x)$
(2)	$\neg\neg\neg B \to \neg\forall x\neg\neg B$	Contraposition Law [L_1, L_2, L_9, MP]
(3)	$\neg B \to \neg\neg\neg B$	[L_1, L_2, L_9, MP]: $A \to \neg\neg A$
(4)	$\neg B \to \neg\forall x\neg\neg B$	Transitivity of implication [L_1, L_2, MP]
(5)	$\exists x\neg B \to \neg\forall x\neg\neg B$	E2-introduction [L_{15}, Gen]
(6)	$\neg\neg\forall x\neg\neg B \to \neg\exists x\neg B$	Contraposition Law [L_1, L_2, L_9, MP]

I-5. In the classical logic, [L_1-L_{11}, L_{13}, L_{14}, MP, Gen]: $\neg\exists x\neg B \to \forall xB$.

(1)	$\neg B \to \exists x\neg B$	Axiom L_{13}: $F(x) \to \exists xF(x)$
(2)	$\neg\exists x\neg B \to \neg\neg B$	Contraposition Law [L_1, L_2, L_9, MP]
(3)	$\neg\neg B \to B$	Classical logic, [L_1-L_{11}, MP]: $\vdash \neg\neg A \to A$

(4) $\neg\exists x\neg B\rightarrow B$ Transitivity of implication [L_1, L_2, MP]

(5) $\neg\exists x\neg B\rightarrow\forall xB$ U2-introduction [L_{14}, Gen]

Thus, we have proved that in Group I, constructively, $F_1\rightarrow F_2\rightarrow F_3\rightarrow F_4\rightarrow F_5$, and, in the classical logic, $F_5\rightarrow F_1$, i.e., we have proved that in Group I: a) in the classical logic, all the formulas are equivalent, and b) constructively, upper formulas imply lower formulas.

I-6. Constructively, [L_1, L_2, L_9, L_{13}, L_{14}, MP, Gen]: $\neg\exists x\neg B\rightarrow\forall x\neg\neg B$.

(1) $\neg B\rightarrow\exists x\neg B$ Axiom L_{13}: $F(x)\rightarrow\exists xF(x)$

(2) $\neg\exists x\neg B\rightarrow\neg\neg B$ Contraposition Law [L_1, L_2, L_9, MP]

(3) $\neg\exists x\neg B\rightarrow\forall x\neg\neg B$ U2-introduction [L_{14}, Gen]

Thus, we have proved that in Group I, constructively, [L_1, L_2, L_9, L_{12} -L_{15}, MP, Gen]: $F_3\rightarrow F_4\rightarrow F_5\rightarrow F_3$, i.e., that formulas F_3, F_4, F_5 are constructively equivalent.

For Group I, it remains to prove

I-7. Constructively, [L_1-L_{10}, MP]: $\neg\neg(\neg\neg\forall xB\rightarrow\forall xB)$.

Immediately, by Theorem 2.5.2(d) [L_1-L_{10}, MP]: $\neg\neg(\neg\neg A\rightarrow A)$.

Group II

II-1. Constructively, [L_1, L_2, L_9, L_{12}-L_{15}, MP, Gen]: $\exists xB\rightarrow\exists x\neg\neg B$.

(1) $B\rightarrow\neg\neg B$ [L_1, L_2, L_9, MP]: $A\rightarrow\neg\neg A$

(2) $\forall x(B\rightarrow\neg\neg B)$ Gen

(3) $\exists xB\rightarrow\exists x\neg\neg B$ Theorem 3.1.2(b) [L_1, L_2, L_{12}-L_{15}, MP, Gen]

II-2. Constructively, [L_1-L_9, L_{12}-L_{15}, MP, Gen]: $\exists x\neg\neg B\rightarrow\neg\neg\exists xB$.

(1) $B\rightarrow\exists xB$ Axiom L_{13}: $F(x)\rightarrow\exists xF(x)$

(2) $\neg\neg B\rightarrow\neg\neg\exists xB$ Theorem 2.4.7(a) [L_1-L_9, MP]: $(A\rightarrow B)\rightarrow(\neg\neg A\rightarrow\neg\neg B)$

(3)	$\forall x(\neg\neg B\rightarrow\neg\neg\exists xB)$	Gen
(4)	$\exists x\neg\neg B\rightarrow\neg\neg\exists xB$	Theorem 3.1.2(b) [L_1, L_2, L_{12}-L_{15}, MP, Gen]

II-3. Constructively, [L_1-L_9, L_{12}-L_{15}, MP, Gen]: $\neg\neg\exists xB\rightarrow\neg\neg\exists x\neg\neg B$.

Immediately from II-1, by Theorem 2.4.7(a) [L_1-L_9, MP]: $(A\rightarrow B)\rightarrow(\neg\neg A\rightarrow\neg\neg B)$.

II-4. Constructively, [L_1-L_9, L_{12}, L_{15}, MP, Gen]: $\neg\neg\exists x\neg\neg B\rightarrow\neg\forall x\neg B$.

(1)	$\forall x\neg B\rightarrow\neg B$	Axiom L_{12}: $\forall xF(x)\rightarrow F(x)$
(2)	$\neg\neg B\rightarrow\neg\forall x\neg B$	Contraposition Law [L_1, L_2, L_9, MP]
(3)	$\exists x\neg\neg B\rightarrow\neg\forall x\neg B$	E2-introduction [L_{15}, Gen]
(4)	$\neg\neg\exists x\neg\neg B\rightarrow\neg\neg\neg\forall x\neg B$	Theorem 2.4.7(a) [L_1-L_9, MP]: $(A\rightarrow B)\rightarrow(\neg\neg A\rightarrow\neg\neg B)$.
(6)	$\neg\neg\neg\forall x\neg B\rightarrow\neg\forall x\neg B$	Theorem 2.4.5 [L_1-L_9, MP]: $\neg\neg\neg A\leftrightarrow\neg A$
(7)	$\neg\neg\exists x\neg\neg B\rightarrow\neg\forall x\neg B$	Transitivity of implication [L_1, L_2, MP]

II-5. In the classical logic, [L_1-L_{11}, L_{13}, L_{14}, MP, Gen]: $\neg\forall x\neg B\rightarrow\exists xB$.

(1)	$\neg\forall x\neg B\rightarrow\neg\neg\exists xB$	II-6 [L_1, L_2, L_9, L_{13}, L_{14}, MP, Gen], see below.
(2)	$\neg\neg\exists xB\rightarrow\exists xB$	Classical logic, [L_1-L_{11}, MP]: $\neg\neg A\rightarrow A$
(3)	$\neg\forall x\neg B\rightarrow\exists xB$	From (1) and (2), by transitivity of implication [L_1, L_2, MP].

Thus, we have proved that in Group II, constructively, $F_1\rightarrow F_2\rightarrow F_3\rightarrow F_4\rightarrow F_5$, and, in the classical logic, $F_5\rightarrow F_1$, i.e., we have proved that in Group II: a) in the classical logic, all the formulas are equivalent, and b) constructively, upper formulas imply lower formulas.

II-6. Constructively, [L_1, L_2, L_9, L_{13}, L_{14}, MP, Gen]: $\neg\forall x\neg B\rightarrow\neg\neg\exists xB$.

(1)	$B\rightarrow\exists xB$	Axiom L_{13}: $F(x)\rightarrow\exists xF(x)$.

(2)	$\neg\exists xB\rightarrow\neg B$	Contraposition Law [L_1, L_2, L_9, MP]
(3)	$\forall x(\neg\exists xB\rightarrow\neg B)$	Gen
(4)	$\neg\exists xB\rightarrow\forall x\neg B$	U2-introduction [L_{14}, Gen]
(5)	$\neg\forall x\neg B\rightarrow\neg\neg\exists xB$	Contraposition Law [L_1, L_2, L_9, MP]

Thus, we have proved that in Group II, constructively, [L_1-L_9, L_{12}-L_{15}, MP, Gen]: $F_3\rightarrow F_4\rightarrow F_5\rightarrow F_3$, i.e., that formulas F_3, F_4, F_5 are constructively equivalent.

II-7. Constructively, [L_1-L_{10}, MP]: $\neg\neg(\neg\neg\exists xB\rightarrow\exists xB)$.

Immediately, by Theorem 2.5.2, [L_1-L_{10}, MP]: $\neg\neg(\neg\neg A\rightarrow A)$.

Thus, constructively, $\neg\neg(F_3\rightarrow F_1)$, and $F_1\rightarrow F_2\rightarrow F_3\rightarrow F_4\rightarrow F_5\rightarrow F_3$. By Theorem 2.4.7(d), [$L_1$-$L_9$, MP] $\neg\neg(A\rightarrow B)$, $\neg\neg(B\rightarrow C)\vdash\neg\neg(A\rightarrow C)$. Thus, in fact, we have proved that in Group II, for all i, j, constructively, $\neg\neg(F_i\rightarrow F_j)$ (a kind of "weak equivalence").

Group III

III-1. Constructively, [L_1, L_2, L_9, MP]: $\exists x\neg B\rightarrow\neg\neg\exists x\neg B$

Immediately, by Theorem 2.4.4 [L_1, L_2, L_9, MP]: $A\rightarrow\neg\neg A$.

III-2. Constructively, [L_1, L_2, L_9, L_{12}, L_{15}, MP, Gen]: $\neg\neg\exists x\neg B\rightarrow\neg\forall x\neg\neg B$.

(1)	$\forall x\neg\neg B\rightarrow\neg\neg\forall x\neg\neg B$	I-3 [L_1, L_2, L_9, MP], see above.
(2)	$\neg\neg\forall x\neg\neg B\rightarrow\neg\exists x\neg B$	I-4 [L_1, L_2, L_9, L_{12}, L_{15}, MP, Gen], see above.
(3)	$\forall x\neg\neg B\rightarrow\neg\exists x\neg B$	Transitivity of implication [L_1, L_2, MP]
(4)	$\neg\neg\exists x\neg B\rightarrow\neg\forall x\neg\neg B$	Contraposition Law [L_1, L_2, L_9, MP]

III-3. Constructively, [L_1-L_9, L_{12}, L_{14}, MP, Gen]: $\neg\forall x\neg\neg B\rightarrow\neg\forall xB$.

(1)	$\forall xB\rightarrow\neg\neg\forall xB$	I-1 [L_1, L_2, L_9, MP], see above.
(2)	$\neg\neg\forall xB\rightarrow\forall x\neg\neg B$	I-2 [L_1-L_9, L_{12}, L_{14}, MP, Gen]

(3) $\forall xB \rightarrow \forall x\neg\neg B$ Transitivity of implication [L_1, L_2, MP]

(4) $\neg\forall x\neg\neg B \rightarrow \neg\forall xB$ Contraposition Law [L_1, L_2, L_9, MP]

III-4. In the classical logic, [L_1-L_{11}, L_{13}, L_{14}, MP, Gen]: $\neg\forall xB \rightarrow \exists x\neg B$.

(1) $\neg\exists x\neg B \rightarrow \forall xB$ I-5: in the classical logic, [L_1-L_{11}, L_{13}, L_{14}, MP, Gen]

(2) $\neg\forall xB \rightarrow \neg\neg\exists x\neg B$ Contraposition Law [L_1, L_2, L_9, MP]

(3) $\neg\neg\exists x\neg B \rightarrow \exists x\neg B$ Classical logic, [L_1-L_{11}, MP]: $\vdash \neg\neg A \rightarrow A$

(4) $\neg\forall xB \rightarrow \exists x\neg B$ Transitivity of implication [L_1, L_2, MP]

Thus, we have proved that in Group III, constructively, $F_1 \rightarrow F_2 \rightarrow F_3 \rightarrow F_4$, and, in the classical logic, $F_4 \rightarrow F_1$, i.e., we have proved that in Group III: a) in the classical logic, all the formulas are equivalent, and b) constructively, upper formulas imply lower formulas.

III-4. Constructively, [L_1, L_2, L_9, L_{13}, L_{14}, MP, Gen]: $\neg\forall x\neg\neg B \rightarrow \neg\neg\exists x\neg B$.

(1) $\neg\exists x\neg B \rightarrow \forall x\neg\neg B$ I-6 [L_1, L_2, L_9, L_{13}, L_{14}, MP, Gen]

(2) $\neg\forall x\neg\neg B \rightarrow \neg\neg\exists x\neg B$ Contraposition Law [L_1, L_2, L_9, MP]

Thus, we have proved that in Group III, constructively, $F_2 \rightarrow F_3 \rightarrow F_2$, i.e., that formulas F_2, F_3 are constructively equivalent.

III-5. Constructively, [L_1-L_{10}, MP]: $\neg\neg(\neg\neg\exists x\neg B \rightarrow \exists x\neg B)$.

Immediately, by Theorem 2.5.2 [L_1-L_{10}, MP]: $\neg\neg(\neg\neg A \rightarrow A)$.

Group IV

IV-1. Constructively, [L_1, L_2, L_9, MP]: $\forall x\neg B \rightarrow \neg\neg\forall x\neg B$.

Immediately, by Theorem 2.4.4 [L_1, L_2, L_9, MP]: $A \rightarrow \neg\neg A$.

IV-2. Constructively, [L_1-L_9, L_{12}-L_{15}, MP, Gen]: $\neg\neg\forall x\neg B \rightarrow \neg\exists x\neg\neg B$.

(1) $\exists x\neg\neg B \rightarrow \neg\forall x\neg B$ From II-2, II-3, II-4 [L_1-L_9, L_{12}-L_{15}, MP, Gen], by transitivity of implication [L_1, L_2, MP].

(2) $\neg\neg\forall x\neg B\rightarrow\neg\exists x\neg\neg B$ Contraposition Law [L_1, L_2, L_9, MP]

IV-3. Constructively, [L_1, L_2, L_9, L_{12}-L_{15}, MP, Gen]: $\neg\exists x\neg\neg B\rightarrow\neg\exists xB$.

(1) $\exists xB\rightarrow\exists x\neg\neg B$ II-1 [L_1, L_2, L_9, L_{12}-L_{15}, MP, Gen]

(2) $\neg\exists x\neg\neg B\rightarrow\neg\exists xB$ Contraposition Law [L_1, L_2, L_9, MP]

IV-4. Constructively, [L_1, L_2, L_9, L_{13}, L_{14}, MP, Gen]: $\neg\exists xB\rightarrow\forall x\neg B$.

(1) $B\rightarrow\exists xB$ Axiom L_{13}: $F(x)\rightarrow\exists xF(x)$

(2) $\neg\exists xB\rightarrow\neg B$ Contraposition Law [L_1, L_2, L_9, MP]

(3) $\neg\exists xB\rightarrow\forall x\neg B$ U2-introduction [L_{14}, Gen]

Thus, we have proved that in Group IV all the formulas are constructively equivalent.

And thus, we have proved the positive part of the legend of Table 3.2. The negative part of the legend asserts that the following (classically provable) formulas cannot be proved constructively:

(1) $\neg\neg\forall xB\rightarrow\forall xB$ See Group I. Simply, an instance of (the non-constructive) $\neg\neg A\rightarrow A$.

(2) $\forall x\neg\neg B\rightarrow\neg\neg\forall xB$ See Group I. **Super-non-constructive**: even $\neg\neg$(2) is non-constructive!

(3) $\neg\neg(\forall x\neg\neg B\rightarrow\neg\neg\forall xB)$ $\neg\neg$(2). See Group I.

(4) $\exists x\neg\neg B\rightarrow\exists xB$ See Group II. Nearly, an instance of (the non-constructive) $\neg\neg A\rightarrow A$.

(5) $\neg\neg\exists xB\rightarrow\exists x\neg\neg B$ See Group II. Stronger than simply non-constructivity of $\neg\neg A\rightarrow A$?

(6) $\neg\neg\exists x\neg B\rightarrow\exists x\neg B$ See Group III. Simply, an instance of (the non-constructive) $\neg\neg A\rightarrow A$.

(7) $\neg\forall xB\rightarrow\neg\forall x\neg\neg B$ See Group III. **Super-non-constructive**: even $\neg\neg$(7) is non-constructive!

(8) $\neg\neg(\neg\forall xB\rightarrow\neg\forall x\neg\neg B)$ $\neg\neg$(7). See Group III.

Still, the most striking (classically provable) non-constructive quantifier implications correspond to existence proofs via *reductio ad absurdum*:

(8)	$\neg\forall x\neg B\rightarrow\exists xB$	$\neg\neg$(8) is constructively provable, but (8) is not, see Group II. If we know how to derive a contradiction from $\forall x\neg B$, then may be, we do not know how to find a particular x such that B.
(9)	$\neg\forall x\neg B\rightarrow\neg\neg\exists x\neg\neg B$	(9) is weaker than (8), but still non-constructive, see Group II. If we know how to derive a contradiction from $\forall x\neg B$, then may be, we do not know how to derive a contradiction from $\neg\exists x\neg\neg B$.
(10)	$\neg\forall xB\rightarrow\exists x\neg B$	Even $\neg\neg$(10) is non-constructive, see Group III. If we know how to derive a contradiction from $\forall xB$, then may be, we do not know how to find a particular x such that $\neg B$.
(11)	$\neg\forall xB\rightarrow\neg\neg\exists x\neg B$	(11) is weaker than (10), but still **super-non-constructive** (i.e., even $\neg\neg$(11) is non-constructive), see Group III. If we know how to derive a contradiction from $\forall xB$, then may be, we do not know how to derive a contradiction from $\neg\exists x\neg B$.

3.3. Proving Formulas Containing Conjunction and Disjunction

Theorem 3.3.1.

a) [L_1-L_5, L_{12}, L_{14}, MP, Gen]: $\forall x(B\wedge C)\leftrightarrow\forall xB\wedge\forall xC$.

b) [L_1, L_2, L_6-L_8, L_{12}, L_{14}, MP, Gen]: $\vdash\ \forall xB\vee\forall xC\rightarrow\forall x(B\vee C)$. The converse formula $\forall x(B\vee C)\rightarrow\forall xB\vee\forall xC$ cannot be true. Explain, why.

Proof. Before proving (a), do Exercise 3.3.1.

Exercise 3.3.1. Prove:

a) [L_3-L_5, L_{12}, MP, Gen]: $\forall x(B\wedge C)\ \vdash\ \forall xB\wedge\forall xC$;

b) [L_3-L_5, L_{12}, MP, Gen]: $\forall xB\wedge\forall xC\ \vdash\ \forall x(B\wedge C)$.

Since, in your first proof, Gen has been applied only to x, which does not

appear as a free variable in $\forall x(B \wedge C)$, then, by Deduction Theorem 2 [L_1, L_2, L_{14}, MP, Gen] we obtain

$$[L_1\text{-}L_5, L_{12}, L_{14}, \text{MP, Gen}]: \quad \forall x(B \wedge C) \rightarrow \forall x B \wedge \forall x C \quad .$$

Similarly, in your second proof, Gen has been applied only to x, which does not appear as a free variable in $\forall x B \wedge \forall x C$, then, by Deduction Theorem 2 [L_1, L_2, L_{14}, MP, Gen] we obtain

$$[L_1\text{- } L_5, L_{12}, L_{14}, \text{MP, Gen}]: \quad \forall x B \wedge \forall x C \rightarrow \forall x(B \wedge C) \quad .$$

Now, by C-introduction [L_5] we obtain the equivalence (a) of Theorem 3.3.1.

Exercise 3.3.2. Use D-elimination to prove (b) of Theorem 3.3.1.

Q.E.D.

Theorem 3.3.2.

a) [L_1-L_8, L_{12}-L_{15}, MP, Gen]: $\exists x(B \vee C) \leftrightarrow \exists x B \vee \exists x C$.

b) [L_1-L_5, L_{13}-L_{15}, MP, Gen]: $\exists x(B \wedge C) \rightarrow \exists x B \wedge \exists x C$. The converse implication $\exists x B \wedge \exists x C \rightarrow \exists x(B \wedge C)$ cannot be true. Explain, why.

Exercise 3.3.3. a) Prove (a→) of Theorem 3.3.2. (Hint: start by assuming $B \vee C$, apply D-elimination, etc., and finish by E2-introduction.)

b) Prove (a←) of Theorem 3.3.2. (Hint: start by proving $B \rightarrow \exists x(B \vee C)$ and $C \rightarrow \exists x(B \vee C)$, apply D-introduction and finish E2-introduction.)

c) Prove (b) of Theorem 3.3.2. (Hint: start by assuming $B \wedge C$, derive $\exists x B \wedge \exists x C$, and finish by E2-introduction.)

3.4. Replacement Theorems

An example: we know that $\log xy = \log x + \log y$. Hence,

$$\log 2^a 3^b = \log 2^a + \log 3^b \quad .$$

The latter formula represents an instance of the former one. But we know also that $\log x^y = y \cdot \log x$, hence:

$$\log 2^a 3^b = \log 2^a + \log 3^b = a \cdot \log 2 + b \cdot \log 3 \quad .$$

Here, we applied the formula $\log x^y = y \cdot \log x$ to replace *sub-formulas* $\log 2^a$ and $\log 3^b$ by $a \cdot \log 2$ and $b \cdot \log 3$. In school algebra, a legal move.

Now, logic: we know the following theorem of the classical logic: $(A \to B) \leftrightarrow \neg A \vee B$. This equivalence could be used to replace implications by negations and disjunctions, for example, in the formula $(X \to Y) \to Z$. The first step is straightforward:

$$(X \to Y) \to Z \leftrightarrow \neg (X \to Y) \vee Z \quad .$$

But the next step, allowing to obtain the formula $\neg(\neg X \vee Y) \vee Z$ means already that the equivalence $(A \to B) \leftrightarrow \neg A \vee B$ is applied to the **sub-formula** $X \to Y$: it was **replaced** by $\neg X \vee Y$.

We know also that $\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$, hence, we could continue, obtaining further formulas $(\neg\neg X \wedge \neg Y) \vee Z$ and $(X \wedge \neg Y) \vee Z$ (since ¬¬A↔A). Again, we replaced **sub-formulas** by equivalent formulas..

But we would wish to conclude that the final result of our transformations is equivalent to the initial formula:

$$(X \to Y) \to Z \leftrightarrow (X \wedge \neg Y) \vee Z \quad .$$

Until now, in our proofs, we **were not allowed** to use this very natural kind of mathematical argument: if we replace sub-formulas of some formula F by equivalent sub-formulas, then we obtain a formula F' that is equivalent to F .

In this section we will prove meta-theorems filling this gap, the so-called **replacement theorems**.

We will prove also that the meaning of a formula does not depend on the names of bound variables used in it. For example,

$$(\exists xB(x) \to C) \leftrightarrow (\exists yB(y) \to C).$$

Note. To prove all these replacement theorems we will need only the **minimal logic** [L_1-L_9, L_{12}-L_{15}, MP, Gen].

Sub-formulas and Occurrences

Intuitively, B is a sub-formula of the formula C, if B is a formula, and B is a part (substring) of C. But note that a sub-formula may appear in the same formula more than once, as, for example, in the following instance of the axiom L_1: $\exists \mathbf{xB(x)} \to (\exists xC(x) \to \exists \mathbf{xB(x)})$. Thus, it would be more correctly to speak about **occurrences** of sub-formulas. In the above example, there are two occurrences of the formula ∃xB(x).

The formal definition is as follows:

a) o(B) is an occurrence of B in B.

b) If o(B) is an occurrence of B in C, then o(B) is an occurrence of B in ¬C, in $C \wedge D$, $D \wedge C$, $C \vee D$, $D \vee C$, C→D, and D→C.

b) If o(B) is an occurrence of B in C, then o(B) is an occurrence of B in ∃xC, and ∀xC.

We can define also the notion of **propositional** occurrences:

a) o(B) is a propositional occurrence of B in B.

b) If o(B) is a propositional occurrence of B in C, then o(B) is a propositional occurrence of B in ¬C, in $C \wedge D$, $D \wedge C$, $C \vee D$, $D \vee C$, C→D, and D→C.

Intuitively, o(B) is a propositional occurrence of B in C, if, in C, no quantifiers stand over o(B).

Replacement Lemma 1. In the minimal logic, [L_1-L_9, MP]:

(a) A↔B ├ (A→C)↔(B→C) [L_1-L_5, MP]

(b) A↔B ├ (C→A)↔(C→B) [L_1-L_5, MP]

(c) A↔B ├ $A \wedge C \leftrightarrow B \wedge C$ [L_1-L_5, MP]

(d) A↔B ├ $C \wedge A \leftrightarrow C \wedge B$ [L_1-L_5, MP]

(e) A↔B ├ $A \vee C \leftrightarrow B \vee C$ [L_1-L_8, MP]

(f) A↔B ├ $C \vee A \leftrightarrow C \vee B$ [L_1-L_8, MP]

(g) A↔B ├ ¬A↔¬B [L_1-L_9, MP]

Proof. To prove (a), we will first prove that [L_1, L_2, L_4, MP]: A↔B ├ (A→C)→(B→C).

(1) $(A \rightarrow B) \wedge (B \rightarrow A)$ A↔B – hypothesis assumed.

(2) A→C Hypothesis assumed.

(3) B→A From (1), by Axiom L_4.

(4) B→C From (3) and (2), by transitivity of implication [L_1, L_2, MP].

Thus, by [L_1, L_2, MP] Deduction Theorem 1, [L_1, L_2, L_4, MP]: A↔B ├ (A→C)→(B→C).

In a similar way, we can prove that

$$[L_1, L_2, L_3, MP]: A \leftrightarrow B \vdash (B \rightarrow C) \rightarrow (A \rightarrow C).$$

Now, by C-introduction $[L_5, MP]$, we obtain (a).

Exercise 3.4.1. Prove (b, c, d) of Replacement Lemma 1.

Exercise 3.4.2. Prove (e, f, g) of Replacement Lemma 1.

Q.E.D.

Replacement Theorem 1. Let us consider three formulas: B, B', C, where B is a sub-formula of C, and o(B) is a **propositional** occurrence of B in C. Let us denote by C' the formula obtained from C by replacing o(B) by B'. Then, in the minimal logic,

$$[L_1\text{-}L_9, MP]: B \leftrightarrow B' \vdash C \leftrightarrow C'.$$

Proof. Induction by the "depth" of the propositional occurrence o(B).

Induction base: depth = 0. Then C is B, and C' is B'. The conclusion is obvious.

Induction step. If C is not B, then one of the following holds:

a) C is $F \rightarrow G$, and o(B) is in F.

b) C is $F \rightarrow G$, and o(B) is in G.

c) C is $F \wedge G$, and o(B) is in F.

d) C is $F \wedge G$, and o(B) is in G.

e) C is $F \vee G$, and o(B) is in F.

f) C is $F \vee G$, and o(B) is in G.

g) C is $\neg F$, and o(B) is in F.

Case (a). By induction assumption, $[L_1\text{-}L_9, MP]$: $B \leftrightarrow B' \vdash F \leftrightarrow F'$. By Replacement Lemma 1(a), $[L_1\text{-}L_9, MP]$: $F \leftrightarrow F' \vdash (F \rightarrow G) \leftrightarrow (F' \rightarrow G)$. Thus,

$$[L_1\text{-}L_9, MP]: B \leftrightarrow B' \vdash C \leftrightarrow C'.$$

Exercise 3.4.3. Repeat the above argument for the remaining cases (b, c, d, e, f, g).

Q.E.D.

Now, we can use the replacement argument mentioned at the beginning of this section – at least, for propositional occurrences of equivalent sub-formulas.

Replacement Lemma 2. In the minimal logic, $[L_1\text{-}L_9, L_{12}\text{-}L_{15}, MP, Gen]$:

(a) $B \leftrightarrow C \vdash \forall x B \leftrightarrow \forall x C$ $[L_1\text{-}L_5, L_{12}, L_{14}, MP, Gen]$

(b) $B \leftrightarrow C \vdash \exists x B \leftrightarrow \exists x C$ $[L_1\text{-}L_5, L_{12}\text{-}L_{15}, MP, Gen]$

Exercise 3.4.4. Prove Replacement Lemma 2.

Replacement Theorem 2. Let us consider three formulas: B, B', C, where B is a sub-formula of C, and o(B) is **any** occurrence of B in C. Let us denote by C' the formula obtained from C by replacing o(B) by B'. Then, in the minimal logic,

$$[L_1\text{-}L_9, L_{12}\text{-}L_{15}, MP, Gen]: B \leftrightarrow B' \vdash C \leftrightarrow C'.$$

Proof. Induction by the "depth" of the occurrence o(B).

Induction base: depth = 0. Then C is B, and C' is B'. The conclusion is obvious.

Induction step. If C is not B, then one of the following holds:

a)-g) – as in the proof of Replacement Theorem 1.

h) C is $\forall x F$, and o(B) is in F.

i) C is $\exists x F$, and o(B) is in F.

Case (h). By induction assumption, $[L_1\text{-}L_9, L_{12}\text{-}L_{15}, MP, Gen]: B \leftrightarrow B' \vdash F \leftrightarrow F'$. By Replacement Lemma 2(a), $[L_1\text{-}L_9, L_{12}\text{-}L_{15}, MP, Gen]: F \leftrightarrow F' \vdash \forall x F \leftrightarrow \forall x F'$. Thus, $[L_1\text{-}L_9, L_{12}\text{-}L_{15}, MP, Gen]: B \leftrightarrow B' \vdash C \leftrightarrow C'$.

Case (i). By induction assumption, $[L_1\text{-}L_9, L_{12}\text{-}L_{15}, MP, Gen]: B \leftrightarrow B' \vdash F \leftrightarrow F'$. By Replacement Lemma 2(b), $[L_1\text{-}L_9, L_{12}\text{-}L_{15}, MP, Gen]: F \leftrightarrow F' \vdash \exists x F \leftrightarrow \exists x F'$. Thus, $[L_1\text{-}L_9, L_{12}\text{-}L_{15}, MP, Gen]: B \leftrightarrow B' \vdash C \leftrightarrow C'$.

Q.E.D.

Now (only now!), we may use in our proofs the replacement argument mentioned at the beginning of this section. And, for any equivalent sub-formulas!

Finally, let us prove that the meaning of a formula does not depend on the names of bound variables used in it. Intuitively, it "must be so", but now we will prove this intuition as a meta-theorem.

Replacement Lemma 3. If the formula B does not contain the variable y, then (in the minimal logic):

a) $[L_5, L_{12}, L_{14}, MP, Gen]: \forall x B(x) \leftrightarrow \forall y B(y)$;

b) $[L_5, L_{13}, L_{15}, MP, Gen]: \exists x B(x) \leftrightarrow \exists y B(y)$.

Proof. a) First, let us prove [L_{12}, L_{14}, MP, Gen]: $\forall$xB(x)→$\forall$yB(y).

(1) $\forall$xB(x)→B(y)	Axiom L_{12}: $\forall$xF(x)→F(t). B(x) does not contain y, hence, B(x/y) is an admissible substitution.
(2) ($\forall$xB(x)→$\forall$yB(y))	By U2-introduction [L_{14}, Gen].

The converse [L_{12}, L_{14}, MP, Gen(x)]: $\forall$yB(y)→$\forall$xB(x) is proved in a similar way. Now, by C-introduction [L_5, MP], we obtain (a).

b) First, let us prove [L_5, L_{13}, L_{15}, MP, Gen]: $\exists$xB(x)→$\exists$yB(y).

(1) B(x)→$\exists$yB(y)	Axiom L_{13}: F(t)→$\exists$yF(y). B(y) does not contain x, hence, B(y/x) is an admissible substitution.
(2) $\exists$xB(x)→$\exists$yB(y)(	By E2-introduction [L_{15}, Gen].

The converse [L_{13}, L_{15}, MP, Gen(x)]: $\exists$yB(y)→$\exists$xB(x) is proved in a similar way. Now, by C-introduction [L_5, MP], we obtain (b).

Q.E.D.

Replacement Theorem 3. Let *y* be a variable that does not occur in a formula F, containing an occurrence of a quantifier $\forall$x (or $\exists$x). Let us replace by *y* all occurrences of the variable *x* bound by this particular quantifier occurrence. Let us denote the resulting formula by F'. Then, in the minimal logic,

$$[L_1\text{-}L_9, L_{12}\text{-}L_{15}, \text{MP, Gen}]: \vdash F \leftrightarrow F'.$$

Proof. Thus, the formula F contains a sub-formula $\forall$xB(x) (or $\exists$xB(x)), and we wish to replace it by $\forall$y(B(y) (or $\exists$yB(y)), where y does not occur in F. By Replacement Lemma 3, in the minimal logic, $\forall$xB(x)↔$\forall$yB(y), and $\exists$xB(x)↔$\exists$yB(y). Hence, by Replacement Lemma 2, in the minimal logic, F↔F'. Q.E.D.

Now let us repeat our example. We know that (in the classical logic): $(A \to B) \leftrightarrow \neg A \vee B$. Hence, the formula (X→Y)→Z **is** equivalent to $\neg(X \to Y) \vee Z$, and to $\neg(\neg X \vee Y) \vee Z$. We know also that $\neg(A \vee B) \leftrightarrow \neg A \wedge \neg B$, hence, we can continue: (X→Y)→Z **is** equivalent to $(\neg\neg X \wedge \neg Y) \vee Z$, and to $(X \wedge \neg Y) \vee Z$ (since ¬¬A↔A).

Now, in our logic, we can use freely this very natural kind of mathematical argument. And we will do that in the subsequent sections.

3.5. Constructive Embedding

Glivenko's Theorem (see Section 2.7) provides a simple "constructive embedding" for the classical propositional logic: any classically provable formula can be "proved" in the constructive logic, if you put two negations before it. This theorem does not hold for the predicate logic. For example (see Section 3.2),

II-5. In the classical logic, [L_1-L_{11}, L_{13}, L_{14}, MP, Gen]: $\neg\forall x\neg B\rightarrow\exists xB$.

The double negation of this formula, i.e., the formula $\neg\neg(\neg\forall x\neg B\rightarrow\exists xB)$ cannot be proved in the constructive predicate logic. Thus, instead of the simple operation $\neg\neg F$, we must search for a more complicated embedding operation.

However,

Exercise 3.5.1 (optional, for smart students). Verify that a formula F is provable in the classical predicate logic if and only if $\neg\neg F$ is provable in the constructive predicate logic plus the following axiom schema: $\forall x\neg\neg B\rightarrow\neg\neg\forall xB$ (the so-called *Double Negation Shift s*chema, see Intuitionistic Logic by Joan Moschovakis in Stanford Encyclopedia of Philosophy.

The first embedding operation was introduced by Andrey Nikolaevich Kolmogorov (1903-1987) in

A.N.Kolmogorov. On the principle tertium non datur. *Matem. sbornik*, 1925, vol.32, pp.646-667 (in Russian).

A quote from A Short Biography of A.N. Kolmogorov by Paul M.B. Vitanyi follows:

"K. got interested in mathematical logic, and in 1925 published a paper in Mathematicheskii Sbornik on the law of the excluded middle, which has been a continuous source for later work in mathematical logic. This was the first Soviet publication on mathematical logic containing (very substantial) new results, and the first systematic research in the world on intuitionistic logic. K. anticipated to a large extent A. Heyting 's formalization of intuitionistic reasoning, and made a more definite correlation between classical and intuitionistic mathematics. K. defined an operation for `embedding' one logical theory in another. Using this – historically the first such operation, now called the `Kolmogorov operation' – to embed classical logic in intuitionistic logic, he proved that application of the law of the excluded middle in itself cannot lead to a contradiction."

See also Kolmogorov Centennial.

We will investigate the following version of an embedding operation: to obtain O(F), in a formula F, put two negations before: a) every atomic formula, b) every disjunction, c) every existential quantifier. More precisely, let us define the following embedding operation O (you may wish to compare it with some other versions possessing similar properties):

Operation O Detlovs [1964]	**Operation K** Kolmogorov [1925]	**Operation O'** Gödel [1933], see Kleene [1952]	**Operation O^o** Gentzen [1936], see Kleene [1952]
If F is an atomic formula, then O(F) is ¬¬F.	K(F) is ¬¬F.	O'(F) is F.	O^o(F) is F.
O(F→G) is O(F)→O(G).	¬¬(K(F)→K(G))	$\neg(O'(F)\wedge\neg O'(G))$	O^o(F)→O^o(G)
$O(F\wedge G)$ is $O(F)\wedge O(G)$.	$\neg\neg(K(F)\wedge K(G))$	$O'(F)\wedge O'(G)$	$O^o(F)\wedge O^o(G)$
$O(F\vee G)$ is $\neg\neg(O(F)\vee O(G))$	$\neg\neg(K(F)\vee K(G))$	$\neg(\neg O'(F)\wedge\neg O'(G))$	$\neg(\neg Oo(F)\wedge\neg Oo(G))$
O(¬F) is ¬O(F).	¬¬¬K(F), or ¬K(F)*	¬O'(F)	¬O^o(F)
O(∀xF) is ∀xO(F).	¬¬∀xK(F)	∀xO'(F)	∀xO^o(F)
O(∃xF) is ¬¬∃xO(F).	¬¬∃xK(F)	¬∀x¬O'(F)	¬∀x¬O^o(F)

(*) By Theorem 2.4.5, [L_1-L_9, MP]: ¬¬¬K(F)↔¬K(F).

For example, let us take the above formula ¬∀x¬B→∃xB. If B is an atomic formula, then

O(¬∀x¬B→∃xB) is ¬∀x¬¬¬B→¬¬∃x¬¬B, i.e., ¬∀x¬B→¬¬∃x¬¬B.

The latter formula is constructively provable (see Section 3.2, Group II).

Lemma 3.5.1. For any formula F, in the classical logic, F↔O(F).

Proof. By induction. Let us remind that [L_1-L_{11}, MP]: ¬¬A ↔ A.

1. Induction base: F is an atomic formula. Then O(F) is ¬¬F. Since [L_1-L_{11}, MP]: ¬¬F↔F, in the classical logic, O(F)↔F.

2. Induction step.

Case 2a: F is $B\vee C$. Then O(F) is $\neg\neg(O(B)\vee O(C))$.

(1)	O(B)↔B	Induction assumption.
(2)	O(C)↔C	Induction assumption.
(3)	$B\vee C\leftrightarrow O(B)\vee C$	From (1), by Replacement Theorem 1.

(4)	$O(B)\vee C\leftrightarrow O(B)\vee O(C)$	From (2), by Replacement Theorem 1.
(5)	$O(B)\vee O(C)\leftrightarrow\neg\neg(O(B)\vee O(C))$	[L_1-L_{11}, MP]: ¬¬A ↔ A.
(6)	$B\vee C\leftrightarrow\neg\neg(O(B)\vee O(C))$, i.e., F↔O(F)	By transitivity of implication.

Case 2b: F is ∃xB. Then O(F) is ¬¬∃xO(B).

(1)	O(B)↔B	Induction assumption.
(2)	∃xB↔∃xO(B)	From (1), by Replacement Theorem 2.
(3)	∃xO(B)↔¬¬∃xO(B)	[L_1-L_{11}, MP]: ¬¬A ↔ A.
(4)	∃xB↔¬¬∃xO(B), i.e., F↔O(F)	By transitivity of implication.

Case 2c: F is B→C.

Case 2d: F is $B\wedge C$.

Case 2e: F is ¬B.

Case 2f: F is ∀xB.

Exercise 3.5.2. Prove (c, d, e, f).

Q.E.D.

Still, the key feature of the formulas having the form O(F) is given in

Lemma 3.5.2. For any formula F, there is a proof of

$$[L_1\text{-}L_9, L_{12}, L_{14}, \text{MP, Gen}]: \neg\neg O(F)\leftrightarrow O(F).$$

Thus, in the minimal logic, we may drop the double negation before O(F) (before an arbitrary formula, we can do this only in the classical logic).

Note. In some textbooks, if ¬¬G↔G can be proved in the constructive logic, then G is called a **stable formula**. Thus, the embedding O(F) is a stable formula for any F.

Proof. [L_1, L_2, L_9, MP]: A→¬¬A. Thus, it remains to prove ¬¬O(F)→O(F). Let us proceed by induction.

1. Induction base: F is an atomic formula. Then O(F) is ¬¬F, and ¬¬O(F)→O(F) is ¬¬¬¬F→¬¬F. Let us remind that [L_1-L_9, MP]: ¬¬¬A↔¬A. Hence, by taking A = ¬F:

[L_1-L_9, MP]: ¬¬¬¬F→¬¬F, and [L_1-L_9, MP]: ¬¬O(F)→O(F).

2. Induction step.

Case 2a: F is $B \vee C$, or ∃xB, or ¬B. Then O(F) is $\neg\neg(O(B) \vee O(C))$, or ¬¬∃xO(B), or ¬O(B). Hence, ¬¬O(F)→O(F) is ¬¬¬G→¬G, where G is $\neg(O(B) \vee O(C))$, or ¬∃xO(B), or O(B). Let us remind that [L_1-L_9, MP]: ¬¬¬A↔¬A. Hence,

[L_1-L_9, MP] ¬¬¬G→¬G, and [L_1-L_9, MP]: ¬¬O(F)→O(F).

Case 2b: F is B→C. Then O(F) is O(B)→O(C). By induction assumption,

[L_1, L_2, L_{12}, L_{14}, MP, Gen]: ¬¬O(B)→O(B), and ¬¬O(C)→O(C).

(1)	¬¬O(C)→O(C)	Induction assumption.
(2)	¬¬(O(B)→O(C))	¬¬O(F) – hypothesis.
(3)	¬¬O(B)→¬¬O(C)	By Theorem 2.4.7(b): [L_1-L_9, MP]: ¬¬(A→B)→(¬¬A→¬¬B).
(4)	O(B)→¬¬O(B)	[L_1, L_2, L_9, MP]: A→¬¬A.
(5)	O(B)→O(C), i.e., O(F)	From (4), (3) and (1), by transitivity of implication [L_1, L_2, MP].

Hence, since Gen is not applied here at all, by Deduction Theorem 1 [L_1, L_2, MP] we obtain that [L_1-L_9, L_{12}, L_{14}, MP, Gen]: ¬¬O(F)→O(F).

Case 2c: F is $B \wedge C$. Then O(F) is $O(B) \wedge O(C)$. By induction assumption,

[L_1, L_2, L_{12}, L_{14}, MP, Gen]: ¬¬O(B)→O(B), and ¬¬O(C)→O(C).

(1)	$\neg\neg(O(B) \wedge O(C))$	¬¬O(F) – hypothesis.
(2)	$\neg\neg O(B) \wedge \neg\neg O(C)$	From (1), by Theorem 2.4.8(a), [L_1-L_9, MP]: $\neg\neg(A \wedge B) \leftrightarrow (\neg\neg A \wedge \neg\neg B)$.
(3)	¬¬O(B)	From (2), by Axiom L_3.
(4)	¬¬O(C)	From (2), by Axiom L_4.
(5)	O(B)	From (3), by induction assumption.

(6)	O(C)	From (4), by induction assumption.
(7)	$O(B)\wedge O(C)$, i.e., O(F)	From (5) and (6), by Axiom L_5.

Hence, since Gen is not applied here at all, by Deduction Theorem 1 [L_1, L_2, MP] we obtain that [L_1-L_9, L_{12}, L_{14}, MP, Gen]: ¬¬O(F)→O(F).

Case 2d: F is ∀xB. Then O(F) is ∀xO(B). By induction assumption, [L_1-L_9, L_{12}, L_{14}, MP, Gen]: ¬¬O(B)→O(B). We must prove that ¬¬∀xO(B)→∀xO(B).

(1)	¬¬∀xO(B)→∀x¬¬O(B)	Section 3.2, I-2: [L_1-L_9, L_{12}, L_{14}, MP, Gen]: ¬¬∀xB→∀x¬¬B
(2)	├ ¬¬O(B)→O(B)	Induction assumption
(3)	├ ∀x(¬¬O(B)→O(B))	By Gen.
(4)	├ ∀x¬¬O(B)→∀xO(B)	From (3), by Theorem 3.1.2(a), [L_1, L_2, L_{12}, L_{14}, MP, Gen]: ∀x(B→C)→(∀xB→∀xC).
(5)	├ ¬¬∀xO(B)→∀xO(B)	From (1) and (4), by transitivity of implication [L_1, L_2, MP].

Q.E.D.

Lemma 3.5.3. If F is one of the (classical) axioms L_1-L_{11}, L_{12}-L_{15}, then, in the constructive logic, [L_1-L_{10}, L_{12}-L_{15}, MP, Gen]: ├ O(F).

Note. The axiom L_{10} will be used in the proof of Lemma 3.5.3 only once – to prove that O(L_{10}) is provable in the constructive logic. But, of course, O(L_{10}) cannot be proved in the minimal logic, hence, in the Lemma 3.5.3, the constructive logic cannot be replaced by the minimal one.

Proof.

Case 1. F (as an axiom schema) does not contain disjunctions and existential quantifiers, i.e., if F is L_1, L_2, L_3, L_4, L_5, L_9, L_{10}, L_{12}, or L_{14}, then O(F) is an instance of the same axiom as F, thus, [F]: ├ O(F). For example, if F is L_1, i.e., B→(C→B), then O(F) is O(B)→(O(C)→O(B)), i.e., O(F) is an instance of the same axiom L_1.

Case 2a. F is L_6: $B\rightarrow B\vee C$. Then O(F) is $O(B)\rightarrow \neg\neg(O(B)\vee O(C))$,

and [[L_1, L_2, L_6, L_9, MP] ├ O(F). Indeed:

(1) $O(B)\to O(B)\vee O(C)$ — Axiom L_6.

(2) $O(B)\vee O(C)\to\neg\neg(O(B)\vee O(C))$ — [L_1, L_2, L_9, MP]: A→¬¬A.

(3) $O(B)\to\neg\neg(O(B)\vee O(C))$ — By transitivity of implication [L_1, L_2, MP].

Case 2b. F is L_7: $C\to B\vee C$. Then O(F) is $O(C)\to\neg\neg(O(B)\vee O(C))$, and [[L_1, L_2, L_7, L_9, MP] ├ O(F). Proof is similar to Case 2a.

Case 2c. F is L_8: $(B\to D)\to((C\to D)\to(B\vee C\to D))$. Then O(F) is

$(O(B)\to O(D))\to((O(C)\to O(D))\to(\neg\neg(O(B)\vee O(C))\to O(D)))$.

(1) ¬¬O(D)→O(D) — By Lemma 3.5.2, [L_1-L_9, L_{12}, L_{14}, MP, Gen]: ¬¬O(F)→O(F).

(2) O(B)→O(D) — Hypothesis.

(3) (O(C)→O(D) — Hypothesis.

(4) $\neg\neg(O(B)\vee O(C))$ — Hypothesis.

(5) $(O(B)\to O(D))\to((O(C)\to O(D))\to(O(B)\vee O(C)\to O(D)))$. Axiom L_8.

(6) $O(B)\vee O(C)\to O(D)$ — By MP.

(7) $\neg\neg(O(B)\vee O(C))\to\neg\neg O(D)$ — From (6), by Theorem 2.4.7(a), [L_1-L_9, MP]: (A→B)→(¬¬A→¬¬B)

(8) ¬¬O(D) — By MP.

(9) O(D) — From (1), by MP.

Hence, since Gen is not applied after hypotheses appear in the proof, by Deduction Theorem 2A [L_1, L_2, L_{14}, MP, Gen] we obtain that [L_1-L_9, L_{12}, L_{14}, MP, Gen] ├ O(F).

Case 2d. F is L_{11}: $B\vee\neg B$. Then O(F) is $\neg\neg(O(B)\vee\neg O(B))$. Let us remind Theorem 2.4.6(b): [L_1-L_9, MP]: $\neg\neg(A\vee\neg A)$. Hence, [L_1-L_9, MP]

$\vdash O(F)$.

Case 2e. F is L_{13}: $F(t)\rightarrow\exists xF(x)$. Then O(F) is $O(F(t))\rightarrow\neg\neg\exists xO(F(x)))$, and $[[L_1, L_2, L_9, L_{13}, MP] \vdash O(F)$. Indeed:

(1)	$O(F(t))\rightarrow\exists xO(F(x))$	Axiom L_{13}.
(2)	$\exists xO(F(x))\rightarrow\neg\neg\exists xO(F(x))$	$[L_1, L_2, L_9, MP]$: $A\rightarrow\neg\neg A$.
(3)	$\vdash O(F(t))\rightarrow\neg\neg\exists xO(F(x))$	By transitivity of implication $[L_1, L_2,$ MP].

Case 2f. F is L_{15}: $\forall x(F(x)\rightarrow G)\rightarrow(\exists xF(x)\rightarrow G)$. Then O(F) is

$$\forall x(O(F(x))\rightarrow O(G))\rightarrow(\neg\neg\exists xO(F(x))\rightarrow O(G)).$$

(1)	$\neg\neg O(G)\rightarrow O(G)$	By Lemma 3.5.2, $[L_1$-L_9, L_{12}, L_{14}, MP, Gen]: $\neg\neg O(F)\rightarrow O(F)$.
(2)	$\forall x(O(F(x))\rightarrow O(G))$	Hypothesis.
(3)	$\neg\neg\exists xO(F(x))$	Hypothesis.
(4)	$\forall x(O(F(x))\rightarrow O(G))\rightarrow (\exists xO(F(x))\rightarrow O(G))$. Axiom L_{15}: $\forall x(F(x)\rightarrow G)\rightarrow(\exists xF(x)\rightarrow G)$.	
(5)	$\exists xO(F(x))\rightarrow O(G)$	By MP.
(6)	$\neg\neg\exists xO(F(x))\rightarrow\neg\neg O(G)$	From (4), by Theorem 2.4.7(a), $[L_1$-L_9, MP]: $(A\rightarrow B)\rightarrow(\neg\neg A\rightarrow\neg\neg B)$
(7)	$\neg\neg O(G)$	By MP.
(8)	$O(G)$	From (1), by MP.

Hence, since Gen is not applied after hypotheses appear in the proof, by Deduction Theorem 2A $[L_1, L_2, L_{14}$, MP, Gen] we obtain that $[L_1$-$L_9, L_{12}, L_{14}, L_{15}$, MP, Gen] $\vdash O(F)$.

Q.E.D.

Theorem 3.5.4. In the classical logic,

$$[L_1\text{-}L_{11}, L_{12}\text{-}L_{15}, MP, Gen]: B_1, B_2, ..., B_n \vdash C$$

if and only if, in the constructive logic,

$$[L_1\text{-}L_{10}, L_{12}\text{-}L_{15}, MP, Gen]: O(B_1), O(B_2), ..., O(B_n) \vdash O(C).$$

In particular, a formula F is provable in the classical logic if and only if the formula O(F) is provable in the constructive logic.

Proof.

1. Let $[L_1\text{-}L_{11}, L_{12}\text{-}L_{15}, MP, Gen]: B_1, B_2, ..., B_n \vdash C$. Induction by the length of the shortest proof.

Induction base. If C is an axiom, then, by Lemma 3.5.3, in the constructive logic, $\vdash O(C)$. If C is B_i, then $O(B_i) \vdash O(C)$ in any logic.

Induction step.

If C is derived by MP from B and B→C, then, by induction assumption, in the constructive logic: $O(B_1), O(B_2), ..., O(B_n) \vdash O(B)$, and $O(B_1), O(B_2), ..., O(B_n) \vdash O(B\to C)$. Let us merge these two proofs. Since O(B→C) is O(B)→O(C), then, by MP, in the constructive logic: $O(B_1), O(B_2), ..., O(B_n) \vdash O(C)$.

If C is $\forall x B(x)$, and is derived by Gen from B(x), then, by induction assumption, in the constructive logic: $O(B_1), O(B_2), ..., O(B_n) \vdash O(B(x))$. Hence, by Gen, in the constructive logic:

$$O(B_1), O(B_2), ..., O(B_n) \vdash \forall x O(B(x)),$$

i.e., $O(B_1), O(B_2), ..., O(B_n) \vdash O(F)$.

Q.E.D.

2. Let in the constructive logic: $O(B_1), O(B_2), ..., O(B_n) \vdash O(C)$. By Lemma 3.5.1, in the classical logic, $B_i\to O(B_i)$ for all i, and O(C)→C. Hence, in the classical logic, $B_1, B_2, ..., B_n \vdash C$.

Q.E.D.

Corollary 3.5.5. If, in the classical logic, $B_1, B_2, ..., B_n \vdash C\wedge\neg C$, then, in the constructive logic, $O(B_1), O(B_2), ..., O(B_n) \vdash O(C)\wedge\neg O(C)$. I.e., if the postulates $B_1, B_2, ..., B_n$ are inconsistent in the classical logic, then the postulates $O(B_1), O(B_2), ..., O(B_n)$ are inconsistent in the constructive logic.

Or: if the postulates $O(B_1)$, $O(B_2)$, ..., $O(B_n)$ are consistent in the constructive logic, then the postulates B_1, B_2, ..., B_n are consistent in the classical logic.

Corollary 3.5.6. If, for some predicate language, the classical logic would be inconsistent, then so would be the constructive logic. Or: **if, for some predicate language, the constructive logic is consistent, then so is the classical logic** (Gödel [1933], Gentzen [1936]).

Note. In Section 4.3, we will prove that the classical predicate logic is consistent for any predicate language. This will make Corollary 3.5.6 obsolete.

Attention! Corollary 3.5.6 does not extend immediately to **first order theories**, having their own specific non-logical axioms. Consistency must be verified separately for each theory! For example,

Exercise 3.5.3 (optional, for smart students). Verify that, if the constructive first order arithmetic is consistent, then so is the classical first order arithmetic (Gödel [1933], Gentzen [1936]). (Hint: verify that, a) atomic formulas of arithmetic are stable – this is the hard part of the proof, b) if F is an axiom of arithmetic, then so is O(F).)

Thus, the **non-constructivity does not add contradictions (at least) to arithmetic**. If it would, then we could derive "constructive" arithmetical contradictions as well.

Kurt Gödel. Zur intuitionistischen Arithmetik und Zahlentheorie. *Ergebnisse eines mathematischen Kolloquiums*, 1933, Vol. 4, pp. 34-38.

Gerhard Gentzen. Die Widerspruchsfreiheit der reinen Zahlentheorie. *Mathematische Annalen*, 1936, Vol. 112, pp. 493-565.

About constructive embedding operations as a general notion see

Nikolai A.Shanin. Embedding the classical logical-arithmetical calculus into the constructive logical-arithmetical calculus. *Dokladi AN SSSR*, 1954, vol. 94, N2, pp.193-196 (in Russian).

4. Completeness Theorems (Model Theory)

4.1. Interpretations and models

In Chapter 1 we formulated the axioms and rules of inference of constructive and classical logic, and in Chapters 2 and 3 we explored many of their consequences. However, did we succeed in formulation of **all** the necessary logical axioms and rules of inference, without any omissions? Is the classical logical [L_1–L_{15}, MP, Gen] "complete"? And, in which sense could be "complete" the constructive logic, in which the formula $B \vee \neg B$ cannot be proved? What could such questions mean precisely?

Moreover, our logics are, in fact, reconstructions, and **not copies** of "rules of correct thinking". Indeed, let us recall that the axioms L_1, L_{10} were introduced to keep the system simple enough, and not because of their "truth". How to define "completeness" of such artificial systems?

In Section 1.3 we introduced the following definition of completeness: a **theory** is called complete, if it proves or disproves any closed formula of its language. Most mathematical theories are not complete in this sense (Gödel's Incompleteness Theorem).

However, this definition does not apply, if we wish to define the completeness of a **logical system** (constructive or classical logic). Applied to a particular predicate language, such a system represents a theory that does not possess any specific axioms. Such an "empty" theory cannot pretend to prove or disprove all the closed formulas of the language. For example, no logical system will be able to prove or disprove the formula $\forall x\,(Male(x) \vee Female(x))$.

In Section 1.3 we formulated the logical axioms and rules of inference to formalize the manipulation of logical connectives and quantifiers. These axioms and rules are intended to be valid for any predicate languages, independently of the "meanings" of the language primitives (object constants, functions, predicates). Hence, we could qualify a logical system as "complete", if this system could prove all the formulas that are valid ("true") for any predicate languages, independently of the "meanings" of the language primitives.

Thus, we must introduce an explicit general definition of "assigning meanings to the language primitives".

An axiom-less predicate language can limit this assigning only in very limited ways.

Example for mathematicians

The formal language of first order arithmetic (language primitives are x, y, ..., 0, 1 +, *, =) can be used not only to discuss natural numbers, but also to discuss any *rings*, such as, for example, $Z_2=\{0,1\}$, with addition and multiplication performed *modulo 2*. Then,

a) variables are thought to range over the set {0, 1} only;

b) the object constant 1 means the object 1, the object constant 0 – the object 0;

c) to the function constant "+" the addition operation in Z_2 is assigned, to the function constant "*" – the multiplication operation in Z_2 ;

d) to the predicate constant "=" the equality predicate for the set {0, 1} is assigned.

Thus,

x	y	$x+y$	$x*y$	$x=y$
0	0	0	0	true
0	1	1	0	false
1	0	1	0	false
1	1	0	1	true

In Z_2 , the simplest axiom of arithmetic: $\neg(0=x+1)$ is false. Indeed, in Z_2 , 1+1=0.

On the other hand, such unusual formulas as

$$x+x=0\,;-x=x\,;x^2=x$$

are true in Z_2 , but they would be false, if, instead of Z_2 , the usual arithmetic of natural numbers would be used to assign “meanings” to language primitives (details below).

In mathematical logic, for axiom-less predicate languages, a particular “assignment of meanings to language primitives” is called an **interpretation**.

As we see, if two different theories share the same language, then we can obtain for this language two different interpretations. Some formulas that are true under one of the interpretations, may become false in the second one, an

vice versa.

Another example

In Section 1.2, in our "language about people" we used four names of people (*Britney, John, Paris, Peter*) as object constants and the following predicate constants:

Male(x) − means "x is a male person";
Female(x) − means "x is a female person";
Mother(x, y) − means "x is mother of y";
Father(x, y) − means "x is father of y";
*Married(*x, y) − means "x and y are married";
x=y − means "x and y are the same person".

Now, let us consider the following interpretation of the language – a specific "small four person world":

a) The domain of interpretation – and the range of variables – is:

D = {br, jo, pa, pe} (no people, four character strings only!).

Interpretations of predicate constants are defined by the following truth tables:

x	Male(x)	Female(x)
br	false	true
jo	true	false
pa	false	true
pe	true	false

x	y	Father(x, y)	Mother(x, y)	Married(x, y)	x=y
br	br	false	false	false	true
br	jo	false	false	false	false
br	pa	false	false	false	false
br	pe	false	false	false	false
jo	br	false	false	false	false
jo	jo	false	false	false	true
jo	pa	false	false	false	false
jo	pe	false	false	false	false

pa	br	false	true	false	false
pa	jo	false	true	false	false
pa	pa	false	false	false	true
pa	pe	false	false	true	false
pe	br	true	false	false	false
pe	jo	true	false	false	false
pe	pa	false	false	true	false
pe	pe	false	false	false	true

As we see, in this interpretation ("in this small world"):

character strings *br* and *pa* are female persons, *jo* and *pe* – male persons;

pe is father of *br* and *jo*; *pa* is mother of *br* and *jo*;

pe and *pa* are married;

equality means equality of character strings.

And in this interpretation ("in this small world") it is true that:

"all people are either males, or females (but not both)", because in this interpretation, the corresponding formulas $\forall x(Male(x) \vee Female(x))$ and $\forall x \neg(Male(x) \wedge Female(x))$ qualify as true.

"all mothers are females": formula $\forall x \forall y(Mother(x, y) \rightarrow Female(x))$ qualifies as true;

"all fathers are married people" (in this interpretation, in this artificial small world, not in the real world!): the corresponding formula

$\forall x \forall y(Father(x, y) \rightarrow \exists z\, Married(x, z))$ qualifies as true.

But, in this interpretation ("in this small world"), it is **not** true that

"every person possess a mother", because the corresponding formula $\forall x \exists y\, Mother(y, x)$ qualifies as false for x=*pe* and x=*pa*.

Exercise 4.1.1. Build another interpretation (a crazy one!) of the above "four people language", under which the following formulas would be true: "some people are both male and female", "there are sexless people", "there are persons married on herself", "there is a person having a sexless father", "there is a person having two mothers".

By introducing **specific non-logical axioms**, i.e., by introducing "theory about people" instead of pure axiom-less "language about people" we could disqualify your crazy interpretation of Exercise 4.1.1 – because, for example, the following axioms of "theory about people" are false under it:

$$\forall x(Male(x) \vee Female(x));$$
$$\forall x \neg(Male(x) \wedge Female(x)).$$

An interpretation of the language of some theory T under which all axioms of T qualify as true, is called a **model of T**. Thus, models of T form a subset of all the possible interpretations of the language of T.

Model theory

Investigation of language interpretations (some of them are models of theories) is called **model theory.** It represents a specific approach to investigation of formal theories.

Note. For mathematicians: model theory is using (up to) the full power of set theory. **In model theory, formal theories are investigated by using set theory as a meta-theory.**

Paul Bernays, in 1958: "As Bernays remarks, syntax is a branch of number theory and semantics the one of set theory." See p. 470 of

Hao Wang. EIGHTY YEARS OF FOUNDATIONAL STUDIES. *Dialectica,* Vol. 12, Issue 3-4, pp. 466-497, December 1958.

In Sections 4.1-4.3 we will develop a **model theory for the classical logic**, and in Section 4.4 – a model theory for the constructive propositional logic.

Now, let us present an explicit general definition of interpretations: of the ways of assigning precise meanings to the language primitives. In the result we will obtain a precise notion of "true formulas". *Formulas become true or false only under interpretations.* Any particular predicate language allows for multiple ways of assigning meanings to its primitives – *multiple interpretations*.

Interpretation of a language – the language-specific part

Let L be a predicate language containing:

(a possibly empty) set of object constants c_1, ..., c_k, ... ,

(a possibly empty) set of function constants f_1, ..., f_m, ..., and

(a non empty) set of predicate constants p_1, ..., p_n, ...

An **interpretation J of the language L** consists of the following two entities (a set and a mapping):

a) A *non-empty finite or infinite* set D_J – the domain of interpretation (it will serve first of all as the range of object variables). (For infinite domains, set theory comes in here.)

b) A mapping int_J that assigns:

- to each object constant c_i – a member $int_J(c_i)$ of the domain D_J (thus, object constants denote particular objects in D_J),

- to each function constant f_i – a function $int_J(f_i)$ from D_J x ... x D_J into D_J (of course, $int_J(f_i)$ possess the same number of arguments as f_i),

- to each predicate constant p_i – a predicate $int_J(p_i)$ on D_J, i.e., a subset of D_J x ... x D_J (of course, $int_J(p_i)$ possess the same number of arguments as p_i).

Thus, in a sense, the mapping int_J assigns "meanings" to the language primitives.

Example. The above interpretation of the “language about people” put in the terms of the general definition:

a) D = {br, jo, pa, pe}.

b) int_J(Britney)=br, int_J(John)=jo, int_J(Paris)=pa, int_J(Peter)=pe.

c) int_J(Male) = {jo, pe}; int_J(Female) = {br, pa}.

d) int_J(Mother) = {(pa, br), (pa, jo)}; int_J(Father) = {(pe, jo), (pe, br)}.

e) int_J(Married) = {(pa, pe), (pe, pa)}.

f) int_J(=) = {(br, br), (jo, jo), (pa, pa), (pe, pe)}.

As the next popular example let us consider the so-called **standard interpretation** S of first order arithmetic PA (it is called also Peano arithmetic):

a) The domain is D_S = {0, 1, 2, ...} − the set of all natural numbers "as we know it" (more precisely – as we define it in set theory).

b) The mapping int_S assigns:

to the object constant 0 (the letter 0) – the number $0 \in D_S$;

to the object constant 1 (the letter 1) – the number $1 \in D_S$;

to the function constant "+" − the function x+y (addition of natural numbers),
to the function constant "*" − the function x*y (multiplication of natural numbers),

to the predicate constant "=" − the predicate x=y (equality of natural numbers).

Yet another interpretation J_1 of the same language:

a) The domain is D_{J1} = {e, a, aa, aaa, ...} − the set of all strings built of the letter "a" (e is the empty string).

b) The mapping int_{J1} assigns: to the object constant 0 – the empty string e, to the object constant 1 – the string "a", to the function constant "+" − the concatenation function of strings, to the function constant "*" − y times concatenation of x, to the predicate constant "=" − the string equality predicate.

Yet another interpretation J_2 (there is no way to disqualify it as a formally correct interpretation of the axiom-less *language* of first order arithmetic):

a) The domain is D_{J2} = {*o*} – a single object *o*.

b) The mapping int_{J2} assigns: to the object constant 0 – the object *o*, to the object constant 1 – the same object *o*, to the function constant "+" − the only possible function f(o, o)=o, to the function constant "*" − the only possible function f(o, o)=o, to the predicate constant "=" − the predicate {(o, o)}.

Some time later, we will introduce specific non-logical **axioms** that will disqualify (at least some of) the above "inadequate" interpretations.

Having an interpretation J of the language L, we can define the notion of **true formulas** (more precisely − the notion of formulas that are **true under the interpretation** J).

As the first step, **terms** of the language L are interpreted as members of D_J or as functions over D_J. Indeed, terms are defined as object constants, or object variables, or their combinations by means of function constants. The term c_i is interpreted as the member $int_J(c_i)$ of D_J, in other words, as a constant (zero-argument) function. The variable x_i is interpreted as the function $X_i(x_i) = x_i$. And, if $t = f_i(t_1, ..., t_q)$, then $int_J(t)$ is defined as the function obtained by substituting the functions $int_J(t_1)$, ..., $int_J(t_q)$ into the function $int_J(f_i)$.

For example (in first order arithmetic), the standard interpretation of the term (1+1)+1 is the number 3, the interpretation of (x+y+1)*(x+y+1) is the function $(x+y+1)^2$.

Important − non-constructivity! Note that, for an infinite domain D_J, the interpretations of function constants are allowed to be **non-computable** functions. However, if these interpretations were all computable (as in the standard interpretation of arithmetic), then we could compute the "value" of any term *t* for any combination of values of variables appearing in *t*.

As the next step, the notion of **true atomic formulas** is defined. Of course, if a formula contains variables (as, for example, the formula x+y=1), then its "truth-value" must be defined for each combination of the values of these variables. Thus, to obtain the truth-value of the formula $p_i(t_1, ..., t_q)$ for some

fixed values of the variables appearing in the terms t_1, .., t_q, we must first take ("compute") the values of these terms, and then substitute these values into the predicate $int_J(p_i)$.

For example (in first order arithmetic), under the standard interpretation S, the formula x+y=1 will be true if and only if either x takes the value 0, and y takes the value 1, or x takes the value 1, and y takes the value 0. Otherwise, this formula is false.

Important − non-constructivity! Note that, for an infinite domain D_J, the interpretations of predicate constants are allowed to be **non-computable** predicates. However, if these interpretations were all computable (as in the standard interpretation of arithmetic), then we could compute the "truth value" of any *atomic* formula F for any combination of values of variables appearing in F.

Interpretations of languages − the standard common part

Finally, we define the notion of **true formulas** of the language L **under the interpretation** J (of course, for a fixed combination of values of their free variables – if any):

a) Truth-values of the formulas $\neg B$, $B \wedge C$, $B \vee C$, $B \rightarrow C$ must be computed from the truth-values of B and C by using the well-known **classical** truth tables (see Section 4.2).

b) The formula $\forall x\, B$ is true under J if and only if B(c) is true under J for all members c of the domain D_J.

c) The formula $\exists x\, B$ is true under J if and only if there is a member c of the domain D_J such that B(c) is true under J.

For example (in first order arithmetic), the formula

$$\exists y((x = y + y) \vee (x = y + y + 1))$$

is intended to say that "*x* is even or odd". Under the standard interpretation S of arithmetic, this formula is true for all values of its free variable *x*. Similarly, $\forall x \forall y(x + y = y + x)$ is a closed formula that is true under this interpretation.

The notion "**a closed formula F is true under the interpretation J**" is now precisely defined.

Attention! What about a formula F(x) containing x as free variable? In principle, under the interpretation J, such formula may be true for some values of x, and false – for some other values. So, if some formula **contains free**

variables, let us say that it is **true** under the interpretation J if and only if it is true **for all combinations** of the values of its free variables.

Important − non-constructivity! It may seem that, under an interpretation, any closed formula is "either true or false". However, note that, for an infinite domain D_J, the notion of "true formulas under J" is extremely **non-constructive**. To establish, for example, the truth-value of the formula $\forall x \forall y(x+y=y+x)$, we must verify the truth of a+b=b+a for infinitely many values of a and b. Of, course, such a verification cannot be performed on a computer. It can only (sometimes) be proved in some theory. The "degree of constructivity" of the formulas like as $\forall x \exists y C(x,y)$, $\forall x \exists y \forall z D(x,y,z)$ etc. is even less...

Empty Domains? Do you think, we should consider also **empty domains** of interpretation? According to the axiom L_{13}: $(B \to B) \to \exists x(B \to B)$, hence, $\exists x(B \to B)$. In an empty domain, this formula would be false. Thus, to cover the case of empty domain, we would be forced to modify our axioms of predicate logic. In this way one obtains the so-called **free logic**. For details, see Free logic in Wikipedia, and Section 2.16 in Mendelson [1997].

Three kinds of formulas

If one explores some formula F of the language L under various interpretations, then three situations are possible:

a) F is **true in all interpretations** of the language L. Formulas of this kind are called **logically valid formulas**.

b) F is **true in some** interpretations of L, and **false − in some** other interpretations of L.

c) F is **false in all interpretations** of L Formulas of this kind are called **unsatisfiable formulas.**

Formulas that are "not unsatisfiable" (formulas of classes (a) and (b)) are called, of course, **satisfiable formulas**: a formula is satisfiable, if it is **true in at least one interpretation**.

Note. These definitions are pretty clear for a **close** formula F. However, remember that, if some formula **contains free variables**, we say that it is **true** under the interpretation J if and only if it is true **for all combinations** of the values of its free variables. This convention allows to apply the above definitions also to the formulas containing free variables. In particular, we will call a formula F, containing free variables, satisfiable, if there in an interpretation under which F is true **for some values** of its free variables.

Theorem 4.1.1. a) F is logically valid if and only if $\neg F$ is unsatisfiable.

b) F is satisfiable if and only if ¬F is not logically valid.

Exercise 4.1.2. a) Verify (a, b) of Theorem 4.1.1.

b) Assume, F contains free variables $x_1, \dots, x_k$. Verify that:

F is logically valid if and only if so is $\forall x_1 \dots \forall x_k F$;

F is satisfiable if and only if so is $\exists x_1 \dots \exists x_k F$.

Logically valid formulas

Some formulas are true under all interpretations, for example:

$$(B \to C) \wedge (C \to D) \to (B \to D) \ ,$$

$$F(x) \to \exists x F(x) \ ,$$

$$\forall x F(x) \to F(x) \ ,$$

$$\forall x (F(x) \to G(x)) \to (\forall x F(x) \to \forall x G(x)) \ ,$$

$$\forall x (F(x) \to G(x)) \to (\exists x F(x) \to \exists x G(x)) \ ,$$

$$\forall x (G(x) \wedge H(x)) \to (\forall x G(x) \wedge \forall x H(x)) \ ,$$

$$\exists x (G(x) \vee H(x)) \to (\exists x G(x) \vee \exists x H(x)) \ .$$

Such formulas are called **logically valid**.

Thus, a logically valid formula is true independently of its "meaning" − the particular interpretations of constants, functions and predicates used in it. But note that here, the (classical!) **"meanings" of propositional connectives and quantifiers remain fixed.**

Hence, in a sense, logically valid formulas are **"content-free"**: being true in all interpretations, they do not provide any specific information about the features of objects they are "speaking" about.

Important − non-constructivity! The notion of logically valid formulas is **doubly non-constructive** in the sense that the universal quantifier "for all interpretations" is added to the (already) non-constructive definition of a formula true in a particular interpretation.

As we will it in Section 4.3, all the axioms of classical logic [L_1–L_{15}, MP, Gen] are logically valid formulas. And it appears also that, from logically valid formulas, the inference rules MP and Gen generate only logically valid formulas. In this way we will prove that **all the formulas that can be proved in the classical logic [L_1–L_{15}, MP, Gen], are logically valid** (the so-called **soundness theorem** of the classical predicate logic, see Section 4.3).

As an example, let us verify in detail that the axiom L_{12}: ∀xF(x)→F(t) (where the substitution F(x/t) is admissible) is logically valid.

Let us assume the contrary: that, under some interpretation J, for some values of its free variables (if any), L_{12} is false. According to the classical truth tables, this could happen only, if ∀xF(x) were true, and F(t) were false (under the interpretation J, for the same above-mentioned values of free variables). Let us "compute" the value of the term t for these values of free variables (since the substitution F(x/t) is admissible, t may contain only these variables), and denote it by c. Thus, F(c) is false. But ∀xF(x) is true, hence, F(a) is true for all $a \in D_J$, i.e., F(c) also is true. Contradiction. Hence, L_{12} is true under all interpretations for all combinations of its free variables (if any).

Another example

Let us verify that the following formula

$$\exists x(G(x) \vee H(x)) \rightarrow (\exists x G(x) \vee \exists x H(x))$$

is logically valid as well. Knowing the soundness theorem, instead of verification, we could “simply” prove this formula in the classical logic. However, the proof is somewhat complicated. The direct verification is here much simpler.

Let us assume that there is an interpretation J such that the formula is false for some values of its free variables, and let us derive a contradiction.

The implication can be false only if the premise $\exists x(G(x) \vee H(x))$ is true, and the conclusion $\exists x G(x) \vee \exists x H(x)$ is false. Thus, there is an object $a \in D_J$ for which G(a) is true, or H(a) is true. In the first case, $\exists x G(x)$ is true, in the second case $\exists x H(x)$ is true. Hence, in both cases $\exists x G(x) \vee \exists x H(x)$ is true. Contradiction. The formula under question is logically valid.

Yet another example

Let us verify that the following formula

$$\forall x(F(x) \rightarrow G(x)) \rightarrow (\forall x F(x) \rightarrow \forall x G(x))$$

is logically valid. Let us assume that there is an interpretation J such that the formula is false for some values of its free variables, and let us derive a contradiction.

The implication can be false only if:

1) the premise $\forall x(F(x) \rightarrow G(x))$ is true, and

2) the conclusion $\forall x F(x) \rightarrow \forall x G(x)$ is false.

3) (2) can be false only if $\forall x F(x)$ is true, and

4) $\forall x G(x)$ is false.

5) (4) means that there is an object $a \in D_J$ for which G(a) is false.

6) (5) means that F(a) is true.

7) (6) and (5) mean that $F(a) \rightarrow G(a)$ is false, hence, (1) is false as well.

Contradiction. The formula under question is logically valid.

Exercise 4.1.3. Verify in detail that the remaining 4 of the above formulas are logically valid as well.

Is our axiom system of logic powerful enough to prove ALL the logically valid formulas? The answer is positive − see **Gödel's Completeness Theorem** in Section 4.3: if a formula is logically valid, then it can be proved in the **classical** predicate logic [L_1–L_{11}, L_{12}–L_{15}, MP, Gen].

But, of course, there are formulas that are not logically valid. For example, negations of logically valid formulas are false in all interpretations, they are cannot be logically valid. Such formulas are called **unsatisfiable formulas**. But there are formulas that are true in some interpretations (are *satisfiable*), and false − in some other ones (are *not logically valid*). Examples of such formulas:

– $\forall x(Male(x) \vee Female(x))$. We already know that this formula can be made true or false by choosing a specific interpretation of the "language about people".

– The axiom of arithmetic $\neg(0 = x+1)$ considered above. It is true under the standard interpretation S, but it is false in the Z_2 - interpretation (because under it, 1+1=0, see above).

To conclude that some formula is **not** logically valid, we must build an interpretation J such that the formula under question is false (for some values of its free variables – if any).

As an example, let us verify that the formula

$$\forall x(p(x) \vee q(x)) \rightarrow \forall x\, p(x) \vee \forall x\, q(x)$$

is **not** logically valid (p, q are predicate constants). Why it is not? Because the truth-values of p(x) and q(x) may behave in such a way that $p(x) \vee q(x)$ is always true, but neither ∀x p(x), nor ∀x q(x) is true. Indeed, let us take the domain D = {a, b}, and set (in fact, we are using one of two possibilities):

x	**p(x)**	**q(x)**
a	true	false
b	false	true

In this interpretation, $p(a)\vee q(a)$ = true, $p(b)\vee q(b)$ = true, i.e., the premise $\forall x(p(x)\vee q(x))$ is true. But the formulas $\forall$x p(x), $\forall$x q(x) both are false. Hence, in this interpretation, the conclusion $\forall x\, p(x)\vee\forall x\, q(x)$ is false, and $\forall x(p(x)\vee q(x))\to\forall x\, p(x)\vee\forall x\, q(x)$ is false. We have built an interpretation, making the formula false. Hence, it is **not logically valid**. Q.E.D.

On the other hand, this formula **is satisfiable** – there is an interpretation under which it is true. Indeed, let us take D={a} as the domain of interpretation, and let us set p(a)=q(a)=true. Then all the formulas

$$\forall x(p(x)\vee q(x)),\ \forall x\, p(x),\ \forall x\, q(x)$$

become true, and so becomes the entire formula. Q.E.D.

Exercise 4.1.4. Verify that the following formulas are satisfiable, but **not** logically valid (p, q, r are predicate constants):

a) $p(x,y)\wedge p(y,z)\to p(x,z)$,

b) q(x)→$\forall$x q(x),

c) ($\forall$x q(x)→$\forall$x r(x))→$\forall$x(q(x)→r(x)),

c_1) $\exists$x(p(x)→B)→($\exists$x p(x)→B), where B does not contain x,

d) $\forall$x$\exists$y p(x, y)→$\exists$y$\forall$x p(x, y),

e) $\exists x\, q(x)\wedge\exists x\, r(x)\to\exists x(q(x)\wedge r(x))$,

f) $\forall x\neg p(x,x)\wedge\forall x\forall y\forall z(p(x,y)\wedge p(y,z)\to p(x,z))\ \to$

$$\forall x\forall y(x=y\vee p(x,y)\vee p(y,x))\ .$$

Hint. For the domain D={a, b}, it is convenient to use *separate* tables to define your interpretations of unary (see above) and binary predicates, for example,

x	**y**	**r(x, y)**
a	a	false
a	b	true
b	a	true
b	b	false

Exercise 4.1.5. Are the following formulas logically valid, or not (p, q are predicate constants):

($\exists$x p(x)→$\exists$x q(x))→$\exists$x(p(x)→q(x));

$$(\exists x\ p(x)\rightarrow\exists x\ q(x))\rightarrow\forall x(p(x)\rightarrow q(x)).$$

Satisfiability

We already know that, in a predicate language L, a formula F is called **satisfiable** if and only if **there is an interpretation** of the language L such that F is true for **some** values of its free variables.

Examples. a) Formula $\exists x\ p(x)$ is, of course, not logically valid, but it is satisfiable, because it is true in the following interpretation J: $D_J=\{b\}$, p(b)=true.

b) Despite their importance, the formulas x+0=x, x+y=y+x etc. are **not** logically valid (consider "crazy" interpretations of equality and/or addition), but they are satisfiable, for example, they are true under the standard interpretation of arithmetic.

A **set of formulas** $A_1, ..., A_n, ...$ **is called simultaneously satisfiable** if and only if there is an interpretation under which the formulas $F_1, ..., F_n, ...$ all are satisfied. If there is no such interpretation, the set is called **unsatisfiable simultaneously** (a particular formula of the set can be satisfiable, but there is no interpretation making true *all* the formulas of it.)

Lemma 4.1.2. a) A finite set of formulas $A_1,\dots,A_n$ is simultaneously satisfiable if and only if the conjunction $A_1\wedge\dots\wedge A_n$ is satisfiable.

b) If $x_1,\dots,x_k$ is the set of all the variables appearing as free in the formulas, $A_1,\dots,A_n$ then the set $A_1,\dots,A_n$ is simultaneously satisfiable if and only if the formula $\exists x_1\dots\exists x_k(A_1\wedge\dots\wedge A_n)$ is satisfiable.

Exercise 4.1.6. Verify (a, b) of Lemma 4.1.2.

Exercise 4.1.7. a) Verify that the formula $\forall x\forall y(p(x)\rightarrow p(y))$ is true in all one-element interpretations (the interpretation domain consists of a single element), but is false in at least one two-element interpretation (p is a predicate constant).

b) Verify that the formula

$$\forall x\forall y\forall z[(p(x)\leftrightarrow p(y))\vee(q(y)\leftrightarrow q(z))\vee(r(z)\leftrightarrow r(x))]$$

is true in all one- and two-element interpretations, but is false in at least one three-element interpretation (p, q, r are predicate constants).

c) Prove that the formula $\exists x\forall y\ F(x, y)$ is logically valid if and only if so is the formula $\exists x\ F(x, g(x))$, where g is a function constant that does not appear in F.

d) Prove that the formula $\forall x\forall y\exists z\ F(x, y, z)$ is satisfiable if and only if so is the formula $\forall x\forall y\ F(x, y, h(x, y))$, where h is a function constant that does not appear in F.

The problem of reasoning revisited

At the beginning of Section 1.3 we asked the question:

"formula G follows from the formulas $A_1, \ldots, A_n$ ",
what exactly does it mean?

Since we wish to teach reasoning to computers, the answer must be absolutely explicit.

At the end of Section 1.3, having formulated the logical axioms and rules of inference, we arrived at a tentative, but an absolutely precise answer to this question (let us call it **Explication 1**): the assertion "formula G follows from the formulas $A_1, \ldots, A_n$ " means that there is a correct proof (a sequence of formulas) that proves

[L_1-L_{15}, MP, Gen]: $A_1, A_2, \ldots, A_n \vdash G$ (if we intend to use the classical logic), or proves

[L_1-L_{10}, L_{12}-L_{15}, MP, Gen]: $A_1, A_2, \ldots, A_n \vdash G$ (if we intend to use the constructive logic).

But now we have a completely new possibility to approach this question: "formula G follows from formulas $A_1, \ldots, A_n$", what does it mean? Doesn't it mean that "if $A_1, \ldots, A_n$ all are true, then G is true"? We have now a formalized notion of "true" as "true under an interpretation". Thus, we can now formalize the notion of consequence as follows (**Explication 2**):

The assertion "G **follows from** $A_1, \ldots, A_n$" means that **G is true under any interpretation, under which $A_1, \ldots, A_n$ are all true**.

Or, equivalently:

The assertion "G **follows from** $A_1, \ldots, A_n$" means that **G is true in any model of theory {$A_1, \ldots, A_n$}** (see below).

Lemma 4.1.3. Formula G is true under any interpretation, under which the formulas $A_1, \ldots, A_n$ all are true, if and only if the formula $A_1 \wedge \ldots \wedge A_n \rightarrow G$ is logically valid.,

Proof. a) Assume, G is true under any interpretation, under which $A_1, \ldots, A_n$ all are true. Consider $A_1 \wedge \ldots \wedge A_n \rightarrow G$ under an arbitrary interpretation J. If one of the formulas A_i is false under J, then the implication is true. If all the formulas A_i are true under J, then so is G and the implication is true. Thus, $A_1 \wedge \ldots \wedge A_n \rightarrow G$ is logically valid. Q.E.D.

b) Assume, $A_1 \wedge ... \wedge A_n \rightarrow G$ is logically valid, and all the formulas A_1, ..., A_n are true under an interpretation J. Since the implication is true under J as well, so is G. Q.E.D.

Lemma 4.1.4. Assume, G is a closed formula. Then,

$$[L_1\text{-}L_{15}, MP, Gen]: A_1, A_2, ..., A_n \vdash G,$$

if and only if

$$[L_1\text{-}L_{15}, MP, Gen]: \quad A_1 \wedge ... \wedge A_n \rightarrow G \quad .$$

Proof. a) →. Let us start with $A_1 \wedge ... \wedge A_n$, obtain by C-elimination the formulas A_1, A_2, ..., A_n, insert here the given proof of G from the hypotheses A_1, A_2, ..., A_n, and apply Deduction Theorem 2 (G is closed formula). Q.E.D.

b) ←. Let us start with the hypotheses A_1, ..., A_n, obtain by C-introduction $A_1 \wedge ... \wedge A_n$, insert here the given proof of $A_1 \wedge ... \wedge A_n \rightarrow G$, and apply MP. Q.E.D.

In Section 4.3 we will prove **Gödel's Completeness Theorem**. It will follow from this theorem that a formula is logically valid if and only if it is provable in the classical predicate logic. Together with the above lemmas, this means that (for a closed formula G):

Formula G is true under any interpretation, under which all the formulas A_1, ..., A_n are true, if and only if $[L_1\text{-}L_{15}, MP, Gen]: A_1, A_2, ..., A_n \vdash G$.

Thus, for the classical predicate logic, **Explication 1 and Explication 2 of the problem of reasoning are equivalent.**

At first sight, Explication 2 could seem preferable, being simpler and looking "more fundamental", than the very complicated (and therefore, seemingly tentative) Explication 1. But, for the classical logic, both explications are equivalent, so, in fact, Explication 1 is equally "fundamental".

Thus, we can speak simply about "G following from $A_1, ..., A_n$ ", and – depending on the situation – switch to the more convenient of the explications.

Transitive predicates and recursion

How about the predicate **Ancestor(x, y)** − "x is an ancestor of y"? Could it be expressed as a formula of our "language about people"? The first idea − let us "define" this predicate recursively:

$\forall x \forall y (Father(x, y) \vee Mother(x, y) \rightarrow Ancestor(x, y))$;
$\forall x \forall y \forall z (Ancestor(x, y) \wedge Ancestor(y, z) \rightarrow Ancestor(x, z))$.

The second rule declares the transitivity property of the predicate. The above two formulas can

be used as **axioms**, allowing to derive essential properties of the predicate Ancestor(x, y). But how about a single formula F(x, y) in the "language about people", expressing that "x is an ancestor of y"? Such a formula should be a tricky combination of formulas *Father(x, y), Mother(x, y)* and x=y. But such a formula is **impossible**! See Transitive closure in Wikipedia, and Theorem 1.2 (and its proof) in:

Carlos Areces. Logic Engineering. The Case of Description and Hybrid Logics. Ph.D. Thesis, University of Amsterdam, Amsterdam, The Netherlands, 2000.

Exercise 4.1.7 (optional, for smart students). Explain the precise meaning of the statement: in the "language about people", formula F(x, y) expresses that "x is an ancestor of y". Such a formula is impossible, but what does it mean?

Theories and their models

If T is a first order theory, and J is an interpretation of its language, and J makes true the specific axioms of T, then J is called a **model of** T.

For non-mathematical people, the term "model of a theory" may seem strange ("upside down"): in other branches of science, theories serve as a basis for building models of natural phenomena, of technical devices etc. But only the term is strange here, the process is the same as in other branches of science: first order theories "generate" their models, and these models can be used for modeling of natural phenomena, of technical devices etc.

Attention! Specific axioms of a *non-empty* first order theory T cannot be logically valid formulas! They cannot be true in all interpretations, they can be true **only in the models** of T. Models of T form a **proper subclass** of all the possible interpretations.

How precisely could the axioms of a theory specify its interpretations? Up to isomorphism? No, it appears that, for example, there are many **non-standard interpretations** making the axioms of **first order** arithmetic true! By means of first order axioms we cannot exclude the existence of “non-standard” natural numbers, following after all “standard” numbers. More in Non-standard model of arithmetic in Wikipedia.

4.2. Completeness of Classical Propositional Logic

Emil Leon Post (1897-1954): "... Post's Ph.D. thesis, in which he proved the completeness and consistency of the propositional calculus described in the *Principia Mathematica* by introducing the truth table method. He then generalised his truth table method, which was based on the two values "true" and "false", to a method which had an arbitrary finite number of truth-values." (According to MacTutor History of Mathematics archive).

First, let us consider the **classical propositional logic**. Here, each formula is built of some “atoms” B_1, B_2, ..., B_n by using propositional connectives only:

$B \wedge C$, $B \vee C$, $\neg B$, $B \rightarrow C$. Our axioms for this logic we represented as axiom schemas L_1-L_{11}, and the *Modus Ponens* rule of inference, in which the letters B, C, D can be replaced by any formulas.

The problem of completeness

Is our list L_1-L_{11} of classical propositional axiom schemas plus the MP rule of inference "complete"? Aren't some necessary axiom schemas missing there? If something necessary would be missing, we should add it to the list of axioms.

This problem was solved by Emil L. Post in 1920. He proved that **if one would add to the system [L_1-L_{11}, MP] any formula that can't be proved from this system, then one would obtain an inconsistent system in which all formulas are provable.** Thus, in fact, nothing is missing in our list of classical propositional axioms.

Post proved his theorem by using the so-called **classical truth tables**. Each propositional atom may take any of two truth-values – *true* and *false*. And, if we already know truth-values of the formulas B, C, then we can use truth tables to compute truth-values of the formulas $B \wedge C$, $B \vee C$, $\neg B$, $B \rightarrow C$.

If B is false, and C is false, then $B \wedge C$ is false.
If B is false, and C is true, then $B \wedge C$ is false.
If B is true, and C is false, then $B \wedge C$ is false.
If B is true, and B is true, then $B \wedge C$ is true.

B	**C**	$B \wedge C$
0	0	0
0	1	0
1	0	0
1	1	1

If B is false, and C is false, then $B \vee C$ is false.
If B is false, and C is true, then $B \vee C$ is true.
If B is true, and C is false, then $B \vee C$ is true.
If B is true, and C is true, then $B \vee C$ is true.

B	**C**	$B \vee C$
0	0	0
0	1	1

1	0	1
1	1	1

If B is false, then ¬B is true.
If B is true, then ¬B is false.

B	**¬B**
0	1
1	0

No problems so far.

If B is false, and C is false, then B→C is what? True? False? But, why?
If B is false, and C is true, then B→C is what? True? False? But, why?
If B is true, and C is false, then B→C is false, of course.
If B is true, and C is true, then B→C is what? Perhaps, not false? Hence, true?

How to answer these questions? If B is false, then B→C possesses no real meaning. And, if we already know that B is true, and C is true, then B→C is not very interesting. But, if a definite "truth-value" for B→C is mandatory in all cases, then we can *greatly simplify the situation* by assuming that B→C is true, except, when B is true, and C is false. Thus:

If B is false, and C is false, then B→C is true.
If B is false, and C is true, then B→C is true.
If B is true, and C is false, then B→C is false.
If B is true, and C is true, then B→C is true.

B	**C**	**B→C**
0	0	1
0	1	1
1	0	0
1	1	1

This definition is equivalent to saying that B→C is true if and only if $\neg(B \wedge \neg C)$ is true.

In this way, having any formula F and some assignment of truth-values to its atoms, we can compute the truth-value of F.

In fact, this setting **represents a specific kind of *interpretations*** – in terms of the above Section 4.1. Our propositional atoms replace the atomic formulas of predicate languages – both (atoms and formulas) are either true or false, and the truth-value of a propositional formula depends on the truth-values of its atoms only. Thus, we are considering, in fact, interpretations with a fixed domain $D=\{true, false\}$, and in every interpretation, each atom possess a definite truth-value – *true* or *false*.

What would happen to some propositional formula F, if we would try out all the possible interpretations: *all the possible combinations* of truth-values of the propositional atoms occurring in F? This corresponds to trying out a formula of a predicate languages in all the possible interpretations. For example, let us consider the axiom L_{10}:

B	**C**	**¬B**	**B→C**	**¬B→(B→C)**
0	0	1	1	1
0	1	1	1	1
1	0	0	0	1
1	1	0	1	1

In this table, **each row represents a different interpretation**.

In general, there are three possibilities:

F takes only true values (F is logically valid);

F takes only false values (F is unsatisfiable);

F takes both of values (F is satisfiable, but not logically valid).

Soundness of the classical propositional logic

Lemma 4.2.1. Under the classical truth tables, all the classical propositional axioms L_1-L_{11} take only true values (are logically valid).

Proof. First, let us verify L_{11}:

B	**¬B**	$B \vee \neg B$
0	1	1
1	0	1

And L_{10} is, in fact, verified in the table above.

Exercise 4.2.1. Verify that the axioms L_1-L_9 take only true values as well.

Q.E.D.

See also:
"Truth Tables" from The Wolfram Demonstrations Project. Contributed by Hector Zenil.

Lemma 4.2.2. If, under the classical truth tables, the formulas B and B→C take only true values, then so does C. I. e. from "always true" formulas, *Modus Ponens* allows deriving only of "always true" formulas.

Proof. Let us assume that, in some interpretation, C takes a false value. In the same interpretation, B and B→C take true values. If B is true, and C is false, then B→C is false. Contradiction. Hence, C takes only true values. Q.E.D.

Note. In the proof of Lemma 4.2.2, only the third row of implication truth table was significant: if B is true, and C is false, then B→C is false!

Theorem 4.2.3 (soundness of the classical propositional logic).

If [L_1-L_{11}, MP]: ├ F, then, under the classical truth tables, F takes only true values (is logically valid).

Proof. By induction, from Lemmas 4.2.1 and 4.2.2.

In particular: the classical propositional logic is **consistent** – in the sense that one cannot prove both [L_1-L_{11}, MP]: ├ G and [L_1-L_{11}, MP]: ├ $\neg G$, for any formula G. Indeed, a formula and its negation cannot both be true.

Completeness of classical propositional logic

How about the converse statement of Theorem 4.2.3: if, under the classical truth tables, formula F takes only true values, then [L_1-L_{11}, MP]: ├ F? **Is our system powerful enough to prove any formula that is taking only true values?** The answer is "yes":

Theorem 4.2.4 (completeness of the classical propositional logic). Assume, the formula F has been built of the formulas B_1, B_2, ..., B_n by using propositional connectives only. If, under the classical truth tables, for any truth-values of B_1, B_2, ..., B_n, formula F takes only true values, then:

a) in the constructive logic,

$$[L_1\text{-}L_{10}, \text{MP}]: \quad B_1 \vee \neg B_1, B_2 \vee \neg B_2, \ldots, B_n \vee \neg B_n \quad \vdash F,$$

b) in the classical logic, [L_1-L_{11}, MP]: ├ F.

Of course, (b) follows from (a) immediately − all the premises $B_1 \vee \neg B_1, B_2 \vee \neg B_2, ..., B_n \vee \neg B_n$ are instances of the axiom L_{11}.

Corollary 4.2.4. The classical propositional logic [L_1-L_{11}, MP] is "complete" also in the sense that if one would add any formula that can't be proved in this logic, then one would obtain a system, in which all formulas are provable, i.e., an inconsistent system.

Proof of the corollary. Indeed, according to the soundness theorem, if some formula F cannot be proved in [L_1-L_{11}, MP], then it takes false value for some combination of truth-values of its atoms. Replace each true atom by the formula A→A, and each false atom – by ¬(A→A). In this way we obtain a formula F' that takes only false values, hence, ¬F' takes only true values, and hence, according to the completeness theorem, it can be proved in [L_1-L_{11}, MP]. Thus, if we would add F to [L_1-L_{11}, MP] as an axiom schema, then, in this system, the formulas F' and ¬F' will be provable, and by L_{10} – any formula will be provable. Q.E.D.

Note. Assume, the formula F is built of atoms B_1, B_2, ..., B_n by using propositional connectives only. If, under the classical truth tables, for any (possible and impossible) truth-values of B_1, B_2, ..., B_n, formula F takes only true values, then F is called a **tautology**. Thus, Theorem 4.2.4 says that **any tautology can be proved in the classical propositional logic**.

Completeness of the classical propositional logic was first proved by Emil L. Post in his 1920 Ph.D. thesis, and published as

E. Post. Introduction to a general theory of elementary propositions. *American Journal of Mathematics*, 1921, vol. 43, pp.163-185.

About the history, see also:

Richard Zach. Completeness before Post: Bernays, Hilbert, and the development of propositional logic. *The Bulletin of Symbolic Logic*, 1999, vol. 5, N3, pp.331-366.

Now, let us prove Theorem 4.2.2. Following an elegant later idea by Laszlo Kalmar we need two simple lemmas before trying to prove the theorem.

L. Kalmar. Über Axiomatisiebarkeit des Aussagenkalküls. *Acta scientiarium mathematicarum (Szeged)*. 1934-35. vol. 7, pp. 222-243.

Lemma 4.2.5. In the constructive logic, one can "compute" the classical truth-values of $\neg B, B \to C, B \wedge C, B \vee C$ as in the Table 4.1.

Note. Thus, to "compute" the classical truth-values, the axiom L_{11} is not necessary!

Proof.

¬B ├ ¬B

Immediately, in any logic.

B ├ ¬ ¬B

By Theorem 2.4.4. [L_1, L_2, L_9, MP]: A→¬¬A.

¬B, C ├ B→C

¬B, ¬C ├ B→C

By axiom L_{10}: ¬B→(B→C) we obtain ¬B ├ B→C. This covers both cases.

B, ¬C ├ ¬(B→C)

This is exactly Theorem 2.4.2(a) [L_1, L_2, L_9, MP].

B, C ├ B→C

By axiom L_1: C→(B→C) we obtain C ├ B→C.

Table 4.1.

Negation	**Implication**	**Conjunction**	**Disjunction**
[ ]: ¬B ├ ¬B	[**L_{10}**, MP]: ¬B, ¬C ├ B→C	[L_1, L_2, L_3, L_9, MP]: ¬B, ¬C ├ $\neg(B\wedge C)$	[L_1-L_4, L_8, L_9, MP]: ¬B,¬C ├ $\neg(B\vee C)$
[L_1, L_2, L_9, MP]: B ├ ¬¬B	[**L_{10}**, MP]: ¬B, C ├ B→C	[L_1, L_2, L_3, L_9, MP]: ¬B, C ├ $\neg(B\wedge C)$	[L_7, MP]: ¬B, C ├ $B\vee C$
	[L_1, L_2, L_9, MP]: B, ¬C ├ ¬(B→C)	[L_1, L_2, L_4, L_9, MP]: B, ¬C ├ $\neg(B\wedge C)$	[L_6, MP]: B, ¬C ├ $B\vee C$
	[L_1, MP]: B, C ├ B→C	[L_5, MP]: B, C ├ $B\wedge C$	[L_6, MP]: B, C ├ $B\vee C$

¬B, ¬C ├ $\neg(B\wedge C)$

¬B, C ├ $\neg(B\wedge C)$

By axiom L_3: $B\wedge C\rightarrow B$ and the Contraposition Law [L_1, L_2, L_9, MP]: (A→B)→(¬B→¬A) we obtain $\neg B\rightarrow\neg(B\wedge C)$, and ¬B ├

$\neg(B \wedge C)$. This covers both cases.

B, ¬C ├ $\neg(B \wedge C)$

By axiom L_4: $B \wedge C \rightarrow C$ and the Contraposition Law [L_1, L_2, L_9, MP]: (A→B)→(¬B→¬A) we obtain $\neg C \rightarrow \neg(B \wedge C)$, and ¬C ├ $\neg(B \wedge C)$.

B, C ├ $B \wedge C$

By axiom L_5: $B \rightarrow (C \rightarrow B \wedge C)$ we obtain B, C ├ $B \wedge C$.

¬B, ¬C ├ $\neg(B \vee C)$

The most complicated case. By de Morgan Laws, in the minimal logic: $\neg B \wedge \neg C \rightarrow \neg(B \wedge C)$. You may wish to verify that even less – [L_1-L_4, L_8, L_9, MP] is sufficient.

¬B, C ├ $B \vee C$

By axiom L_7: $C \rightarrow B \vee C$ we obtain C ├ $B \vee C$.

B, ¬C ├ $B \vee C$; B, C ├ $B \vee C$

By axiom L_6: $B \rightarrow B \vee C$ we obtain B ├ $B \vee C$. This covers both cases. Q.E.D.

As the next step, let us generalize Lemma 4.2.5 by showing how to "compute" truth-values of arbitrary formula F, which is built of formulas B_1, B_2, ..., B_n by using more than one propositional connective. For example, let us take the formula $B \vee C \rightarrow B \wedge C$:

B	C	$B \vee C$	$B \wedge C$	$B \vee C \rightarrow B \wedge C$
0	0	0	0	1
0	1	1	0	0
1	0	1	0	0
1	1	1	1	1

We will show that, in the constructive logic [L_1-L_{10}, MP]:

¬B, ¬C ├ $B \vee C \rightarrow B \wedge C$,
¬B, C ├ $\neg(B \vee C \rightarrow B \wedge C)$,
B, ¬C ├ $\neg(B \vee C \rightarrow B \wedge C)$,
¬B, ¬C ├ $B \vee C \rightarrow B \wedge C$.

Lemma 4.2.6. Assume, the formula F has been built of the formulas B_1, B_2, ..., B_n by using propositional connectives only. Assume that, if the formulas B_1, B_2, ..., B_n take the truth-values v_1, v_2, ..., v_n respectively, then, for these values, formula F takes the truth-value w. Then, in the constructive logic, we can "compute" the truth-value of F in the following sense:

$$[L_1\text{-}L_{10}, MP]: v_1B_1, v_2B_2, ..., v_nB_n \vdash wF,$$

where: wF denotes F, if w is true, and ¬F, if w is false, and v_iB_i denotes B_i, if v_i is true, and $\neg B_i$, if v_i is false.

Proof. By induction.

Induction base. F is one of the formulas B_i. Then $w=v_i$, and, of course, in any logic, $v_iB_i \vdash wF$.

Induction step.

Note that Lemma 4.2.5 represents the assertion of Lemma 4.2.6 for formulas built of B_1, B_2, ..., B_n by using a single propositional connective.

1. **F is ¬G**. By the induction assumption,

$$[L_1\text{-}L_{10}, MP]: v_1B_1, v_2B_2, ..., v_nB_n \vdash w'G,$$

where w' represents the truth-value of G. By Lemma 4.2.5,

$[L_1\text{-}L_{10}, MP]: w'G \vdash wF$, hence, $[L_1\text{-}L_{10}, MP]: v_1B_1, v_2B_2, ..., v_nB_n \vdash wF$.

2. **F is GoH**, where o is implication, conjunction, or disjunction. By the induction assumption,

$$[L_1\text{-}L_{10}, MP]: v_1B_1, v_2B_2, ..., v_nB_n \vdash w'G,$$

where w' represents the truth-value of G, and

$$[L_1\text{-}L_{10}, MP]: v_1B_1, v_2B_2, ..., v_nB_n \vdash w''H,$$

where w'' represents the truth-value of H. By Lemma 4.2.5,

$$[L_1\text{-}L_{10}, MP]: w'G, w''H \vdash wF,$$

hence, $[L_1\text{-}L_{10}, MP]: v_1B_1, v_2B_2, ..., v_nB_n \vdash wF$.

Q.E.D.

Proof of Theorem 4.2.4(a). By Lemma 4.2.6:

$$[L_1\text{-}L_{10}, MP]: B_1, v_2B_2, ..., v_nB_n \vdash F,$$
$$[L_1\text{-}L_{10}, MP]: \neg B_1, v_2B_2, ..., v_nB_n \vdash F,$$

because F takes only true values. Then, by [L_1, L_2, L_8, MP] D-elimination theorem:

$$[L_1\text{-}L_{10}, MP]: B_1 v \neg B_1, v_2 B_2, ..., v_n B_n \vdash F.$$

By repeating this operation we obtain Theorem 4.2.4(a):

$$[L_1\text{-}L_{10}, MP]: \quad B_1 \vee \neg B_1, B_2 \vee \neg B_2, ..., B_n \vee \neg B_n \quad \vdash F.$$

Q.E.D.

The above proof of the Completeness Theorem contains, in fact, an **algorithm** allowing to build a (very long) proof of any propositional formula that takes only true values.

Computational complexity of the problem

From now on, in principle, we could forget our great ability of proving formulas in the classical propositional logic that we developed in Section 2. Indeed, in order to verify, is a formula provable in [L_1-L_{11}, MP], or not, we can simply check, under the classical truth tables, takes this formula only true values, or not. If it takes, then, by the Completeness Theorem, formula is provable, if not – it is not provable.

For example, instead of trying to prove Peirce's Law *a la* Section 2, we can simply verify that it takes only true values:

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

Thus, by Theorem 4.2.4 (completeness of the classical propositional logic),

$$[L_1\text{-}L_{11}, MP]: \quad ((A \to B) \to A) \to A \quad .$$

For formulas containing one, two or three atoms, this method is really simpler than direct proving of formulas in [L_1-L_{11}, MP]. However, for longer formulas, this method allows for building automatically (by using Lemmas 4.2.5 and 4.2.6) only of extremely long proofs.

Indeed, if the formula F contains *n* different atoms A, B, C, ..., then its truth

table contains 2^n rows for which the truth values of F must be checked one by one. Of course, if the formula contains 2 atoms (like as Peirce's Law, or 3 atoms (like as the Axiom L_2), then its truth table consists of 4 or 8 rows − for most people this is a feasible task. But the truth table for a formula containing 100 atoms contains $2^{100} \approx 10^{30}$ rows to check!

So, let us try inventing a more efficient algorithm? Unfortunately, until now, all the attempts have produced only algorithms the running time of which (for checking of formulas of length n, as $n \to \infty$), exceeds any power n^k .

And it seems, we will never really succeed, because of the Cook-Levin Theorem proved in 1971.

Cook-Levin Theorem. The problem of determining, is a propositional formula **satisfiable**, or not, belongs to the complexity class NP-complete.

For details, see Cook-Levin Theorem, Stephen Cook and Leonid Levin in Wikipedia.

Corollary. The problem of determining, is a propositional formula **provable in the classical propositional logic**, or not, belongs to the complexity class co-NP-complete.

Indeed, a propositional formula is provable in the classical propositional logic if and only if it takes only true values.

It is believed that the running time of any algorithm solving problems of these two classes (for checking of formulas of length n, as $n \to \infty$), exceeds any power n^k . **It is believed, but not yet proved** (see P versus NP problem in Wikipedia) – it is one of the "major unsolved problems in computer science".

The problem of determining the **constructive provability of propositional formulas** is believed to be even harder – it belongs to the complexity class PSPACE-complete, as proved by Richard Statman in 1979:

R. Statman. Intuitionistic propositional logic is polynomial-space complete, Theoretical Computer Science 9 (1979), pp. 67–72 (online copy available).

It is believed that PSPACE-complete problems are strictly harder than co-NP-complete problems.

Worst case estimates and practical experience

When trying to assess the performance of some algorithm, for example, of an algorithm A, which is determining correctly the provability of formulas, we can start by considering of some abstract functions, such as:

$TIME_A(F)$ – the time that A is spending to process the formula F,

$SPACE_A(F)$ – the memory space that A is using to process F.

Then, the “overall performance” of A can be estimated as follows:

$maxTIME_A(n)$ – the time that A is spending to process the “worst” of the formulas having length n;

$maxSPACE_A(n)$ – similarly, for the memory space.

For example, when analyzing the simplest algorithms, that are determining correctly the provability of formulas in the classical propositional logic, one can verify easily that

$$maxTIME_A(n) = 2^{Cn} \quad ,$$

where C is a numerical constant (the less constant, the better the algorithm).

However, this does not imply that all such algorithms are useless in solving of practical problems. The above estimate represents the so-called **worst case estimate:** there exist, indeed, formulas of length n, the processing time of which by the algorithm is unacceptable. But how about the *majority* of formulas? Or, about *specific types* of formulas appearing when solving a specific practical problem? Experience shows that some of the universal algorithms showing unacceptable worst case results, perform really good in many practical cases. For examples, see the tableaux method and the resolution method considered in Section 6 and Section 7.

4.3. Classical Predicate Logic − Gödel's Completeness Theorem

Kurt Gödel (1906-1978) "He is best known for his proof of Gödel's Incompleteness Theorems. In 1931 he published these results in *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme* . ...Gödel's results were a landmark in 20th-century mathematics, showing that mathematics is not a finished object, as had been believed. It also implies that a computer can never be programmed to answer all mathematical questions." (According to MacTutor History of Mathematics archive).

As David Hilbert and Wilhelm Ackermann published in

D. Hilbert, W. Ackermann. Grundzüge der theoretischen Logik. Berlin (Springer), 1928

their, in a sense, "final" version of the axioms of classical logic, they noted: "Whether the system of axioms is complete at least in the sense that all the logical formulas which are correct for each domain of individuals can actually be derived from them, is still an unsolved question."

(quoted after
S. C. Kleene. The Work of Kurt Gödel. "The Journal of Symbolic Logic", December 1976, Vol.41, N4, pp.761-778
See also: Hilbert and Ackermann's 1928 Logic Book by Stanley N. Burris).

Indeed, as we will verify below,

a) all axioms of the classical logic (L_1-L_{15}) are logically valid,

b) the inference rules MP, Gen, from logically valid formulas, infer only logically valid formulas.

Hence, in the classical logic, only logically valid formulas can be proved (soundness theorem). Still, is our list of logical axioms and rules of inference complete in the sense that **all logically valid formulas can be proved?** − the question asked by Hilbert and Ackermann in 1928. The answer is "yes" − as Kurt Gödel established in 1929, in his doctoral dissertation "Über die Vollständigkeit des Logikkalküls" (visit Gödel's archive at the Princeton University Library). The corresponding paper was published in 1930:

K. Gödel. Die Vollständigkeit der Axiome des logischen Funktionenkalküls. "Monatshefte fuer Mathematik und Physik", 1930, Vol.37, pp.349-360.

Theorem 4.3.1 (Gödel's Completeness Theorem). In any predicate language, if a formula is logically valid, then it can be proved by using the classical logic [L_1-L_{15}, MP, Gen].

In fact, a more general theorem can be proved:

Theorem 4.3.2 (Thanks to Sune Foldager for the idea.). If T is a first order theory **with classical logic**, then, if some formula F is true in all models of T, then T proves F.

Thus (as noted in Section 1.3), in principle, we will never need to introduce *specific (non-logical) rules of inference*. All the consequences of the axioms of a first order theory can be derived by using our logical axioms and two rules of inference – *Modus Ponens* and *Generalization*.

Gödel's Completeness Theorem follows from Theorem 4.3.2 (simply consider a theory T with an empty set of specific axioms).

Soundness of the classical predicate logic

First, let us prove the **Soundness Theorem** − that all the formulas that can be proved by using the classical logic [L_1-L_{15}, MP, Gen] are logically valid.

Lemma 4.3.1. All the axioms of the classical logic (L_1-L_{15}) are logically valid.

Proof.

1) Under the classical truth tables, the propositional axioms L_1-L_{11} take only true values (Lemma 4.2.1). Hence, these axioms are true under all interpretations.

2a) $\mathbf{L_{12}}$: ∀xF(x)→F(t), where F is any formula, and t is a term such that the substitution F(x/t) is admissible.

We proved the logical validity of this axiom in Section 4.1.

2b) $\mathbf{L_{13}}$: F(t)→∃xF(x), where F is any formula, and t is a term such that the substitution F(x/t) is admissible.

Similarly, do Exercise 4.3.1.

2c) $\mathbf{L_{14}}$: ∀x(G→F(x))→(G→∀xF(x)), where F is any formula, and G is a formula that does not contain x as a free variable.

Let us assume that, under some interpretation J, for some values of its free variables, L_{14} is false. According to the classical truth tables, this could be only, if ∀x(G→F(x)) were true, and G→∀xF(x) were false (under the interpretation J, for the same above-mentioned values of free variables)

If ∀x(G→F(x)) is true, then G→F(c) is true for all $c \in D_J$. Since G does not contain x, this means that if G is true, then F(c) is true for all $c \in D_J$.

On the other hand, if G→∀xF(x) is false, then G is true, and ∀xF(x) is false. And finally, if ∀xF(x) is false, then F(c) is false for some $c \in D_J$. But, as we established above, if G is true, then F(c) is true for all $c \in D_J$. Contradiction. Hence, under all interpretations, L_{14} is true for all combinations of its free variables.

2d) $\mathbf{L_{15}}$: ∀x(F(x)→G)→(∃xF(x)→G), where F is any formula, and G is a formula that does not contain x as a free variable.

Similarly, do Exercise 4.3.1.

Q.E.D.

Exercise 4.3.1. Verify that the axioms L_{13} and L_{15} are logically valid as well.

Lemma 4.3.2. a) For any interpretation J, from formulas true under J, the inference rules MP and Gen allow deriving only of formulas true under J.

b) From logically valid formulas, the inference rules MP and Gen allow deriving only of logically valid formulas.

Proof. Of course, (b) follows from (a). Let us prove (a).

1. *Modus Ponens.* Assume, B and B→C are true under J. By MP, we derive C. Assume, C is false under J, for some values of its free variables. For these values of the free variables of C, the formulas B and B→C are true under J. Then, according to the classical truth tables, C also must be true. Contradiction. Hence, C is true under J.

2. *Generalization*. Assume, F(x) is true under J, but ∀xF(x) is false under J, for some values of its free variables. Hence, under J, for these values of free variables of ∀xF(x), there is $c \in D_J$ such that F(c) is false. But F(x) is true under J, i.e., F(c) is true as well. Contradiction. Hence, ∀xF(x) is true under J.

Q.E.D.

Theorem 4.3.3 (soundness of the classical predicate logic). All the formulas that can be proved by using the classical logic [L_1-L_{15}, MP, Gen], are logically valid.

Proof. Immediately, by Lemmas 4.3.1 and 4.3.2(b).

Corollary 4.3.3. If T is a first order theory with the classical logic, the specific axioms of which are true under some interpretation J, then:

a) **All theorems** of T **are true** under J as well. In other words, theorems of T are true in any model of T.

b) T is **consistent** (i.e., it does not prove contradictions).

Proof. a) In T, each proof consists of instances of logical axioms (true under any interpretation, Lemma 4.3.1), of instances of specific axioms of T (true under J), and of formulas that can be obtained from these by (repeated) application of MP and Gen. By Lemma 4.3.2(a), the latter are true under J as well.

b) Assume, T proves a contradiction – some formula B and simultaneously – $\neg B$. Then, by (a) these formulas both are true under J. This is impossible.

Q.E.D.

Completeness of the classical predicate logic

The following 12 pages present a proof of Gödel's Completeness Theorem.

Model Existence Theorem

Gödel's Completeness Theorem (and Theorem 4.3.2) are easy consequences of the so-called Model Existence Theorem (see below), first proved in the above-mentioned Gödel's paper of 1930.

Gödel's initial proof was simplified in 1947, when Leon Henkin presented in his Ph.D. thesis a new proof of the Model Existence Theorem. The result was published in 1949:

L. Henkin. The completeness of the first-order functional calculus. "J. Symbolic Logic", 1949, vol.14, pp.159-166.

See also Henkin's later account of his discovery:

L. Henkin. The discovery of my completeness proofs. "The Bulletin of Symbolic Logic", 1996, vol.2, N2, pp.127-158.

An even simpler version Henkin's proof was found independently and almost simultaneously by Gisbert Hasenjäger (during WWII, "was responsible for the security of the Enigma machine"). However, when publishing, Hasenjäger acknowledged Henkin's priority:

G. Hasenjäger. Eine Bemerkung zu Henkin's Beweis für die Vollständigkeit des Prädikatenkalküls der ersten Stufe. "J. Symbolic Logic", 1953, vol.18, pp.42-48.

If T is an inconsistent theory, then there are no models of T. Indeed, if T proves a contradiction, i.e., a formula of the kind $B \wedge \neg B$, then, in a model of T, the formula B must be true and false simultaneously. This is impossible.

Hence, if there is a model of T, then T is consistent.

The converse question: could it be possible that T is a consistent theory, but there are no models of T? The answer is given in the

Theorem 4.3.4 (Model Existence Theorem). If a first order formal theory with classical logic is consistent (in the sense that, by using the classical logic, it does not prove contradictions), then there is a finite or countable model of this theory (an interpretation with a finite or countable domain, under which all axioms and theorems of theory are true).

In the 1920s, some people insisted that mere consistency of a theory (in the syntactic sense of the word − as the lack of contradictions) is not sufficient to regard it as a meaningful theory − as a "theory of something". Model Existence Theorem says the contrary − (syntactic!) consistency of a theory **is** sufficient: **if a theory does not contain contradictions, then it is a "theory of something"** − it describes at least some kind of "mathematical reality". For example, you may think that Euclidean geometry is "meaningless" − because it does not describe 100% correctly the spacial properties of the Universe. But it's your problem, not Euclid's − use another theory, if necessary. Euclidean geometry describes its own kind of "mathematical reality" – and 100% correctly!

Let us assume the Model Existence Theorem (we will prove it later in this Section).

Proof of Theorem 4.3.2.

If T proves F, then F is true in all models of T (Corollary 4.3.3).

Now, let us assume that some formula F is true in all models of theory T (for all values of its free variables, if any), yet it cannot be proved in T. Let us

consider the theory T' in the language of T which contains (besides the axioms of T) an additional non-logical axiom − the negation of F, i.e., the formula $\neg\forall x_1...\forall x_n F$, where x_1, .., x_n are exactly all the free variables appearing in F (if F contains free variables x_1, .., x_n, then, to negate its assertion, we must add the quantifiers $\forall x_1...\forall x_n$). Since F cannot be proved from the axioms of T, **T' is a consistent theory.**

Indeed, if T' would be inconsistent, i.e., we could prove in T' some formula C and its negation ¬C, then we had proofs of [T]: $\neg\forall x_1...\forall x_n F \vdash C$, and [T]: $\neg\forall x_1...\forall x_n F \vdash \neg C$. Since $\neg\forall x_1...\forall x_n F$ is a closed formula, by Deduction Theorem 2, [T]: $\vdash \neg\forall x_1...\forall x_n F \rightarrow C$, and [T]: $\vdash \neg\forall x_1...\forall x_n F \rightarrow \neg C$. Now, by axiom L_9: $(B\rightarrow C)\rightarrow(B\rightarrow\neg C)\rightarrow\neg B$, we obtain that [T]: $\vdash \neg\neg\forall x_1...\forall x_n F$. By the (classical) Double Negation Law, this implies [T]: $\vdash \forall x_1...\forall x_n F$, and by axiom L_{12}: $\forall x B(x)\rightarrow B(x)$ we obtain [T]: $\vdash F$. But, by our assumption, F cannot be proved in T. Hence, T' is a consistent theory, indeed.

Now, by the Model Existence Theorem, there is a model of T' – an interpretation J that makes all its axioms true. Under this interpretation, all axioms of T are true, thus, J is a model of T. And the formula $\neg\forall x_1...\forall x_n F$ (as an axiom of T') is true under J as well. On the other hand, since F is true in all models of T, it is true also under the interpretation J. Hence, formulas $\forall x_1...\forall x_n F$ and $\neg\forall x_1...\forall x_n F$ both are true under J. This is impossible, hence, F must be provable in T. Q.E.D.

Notes. a) Such a simple proof seems almost impossible! We are proving that the logical axioms and rules of inference **are strong enough** to prove all the formulas that are true in all models of T, but where come these axioms in? They come in − in the proof of the Model Existence Theorem. This theorem says that if some formal theory T does not have models, then the logical axioms and rules of inference are strong enough to derive a contradiction from the axioms of T. But the proof of the Model Existence Theorem that we will consider below, is positive, not negative!

b) The above simple proof seems to be extremely non-constructive! "If F is true in all models of T, then it can be proved in T". How could we obtain this proof? Still, how do we know that F is true in all models of T? Only, if we had a constructive procedure that is verifying this, we could ask for an algorithm converting such procedures into proofs in T!

Exercise 4.3.2 (optional, for smart students). Prove the Model Existence Theorem by using the following smart ideas due to Gödel, Henkin and Hasenjäger. Let T be a consistent theory. We must build a model of T. What kind of "bricks" could we use for this "building"? **Idea #1:** let us use *object*

constants of the language! So, let us add to the language of T an infinite set of new object constants d_1, d_2, d_3, ... (and adopt the corresponding additional instances of logical axioms). Prove that this extended theory T_0 is consistent. The model we are building must contain all "objects" whose existence can be proved in T_0. **Idea #2:** for each formula F of T_0 having exactly one free variable (for example, x) let us add to the theory T_0 the axiom $\exists xF(x) \rightarrow F(d_i)$, where the constant d_i is unique for each F. If T_0 proves $\exists xF(x)$, then this constant d_i will represent in our model the object x having the property F. Prove that this extended theory T_1 is consistent. **Idea #3:** prove the (non-constructive) Lindenbaum's Lemma: the axiom set of any consistent theory can be extended in such a way, that the extended theory is consistent and complete (the axiom set of this extended theory may be not algorithmically solvable, let us allow this). By using this lemma, extend T_1 to a consistent *complete* theory T_2. **Idea #4:** let us start building of a model M of T by taking as the domain of M the set of all those terms of T_0 that do not contain variables. And let us interpret each function constant f as the "syntactic constructor function" f', i.e., let us define the value $f'(t_1, ..., t_n)$ simply as the character string "$f(t_1, ..., t_n)$". Finally, let us interpret each predicate constant p as a (non-constructive) relation p' such that $p'(t_1, ..., t_n)$ is true in M if and only if T_2 proves $p'(t_1, ..., t_n)$. To complete the proof, prove that an arbitrary formula G is true in M if and only if T_2 proves G. Hence, all theorems of the initial theory T are true in M.

Adolf Lindenbaum (1904-1941), his wife Janina Hosiasson-Lindenbaum (1899-1942).

Lemma 4.3.4 (Lindenbaum's Lemma). Any consistent first order theory can be extended to a consistent complete theory. More precisely, if T is a consistent first order theory, then, in the language of T, there is a set A of closed formulas such that T+A is a consistent complete theory.

Note. a) By T+A we denote the first order theory in the language of T, obtained from T by adding the formulas of the set A as non-logical axioms.

b) In general, T+A is not a formal theory in the sense of Section 1.1 – the set A not always is algoritmically solvable.

Exercise 4.3.3. Verify that, in any predicate language L, only countably many formulas can be generated: produce an algorithm for printing out a sequence F_0, F_1, F_2, ... containing all the formulas of L.

Proof of Lindenbaum's Lemma (**Attention: non-constructive reasoning!)**

Let us use the algorithm of the Exercise 4.3.3 printing out the sequence F_0, F_1, F_2, ... of all formulas in the language of T, and let us run through this sequence, processing only those formulas F_i that are closed.

At the very beginning, the set of new axioms A_0 is empty.

At the step i, we already have some set A_{i-1} of new axioms. If the formula F_i is not closed, let us ignore it, and set $A_i=A_{i-1}$. Now, let us suppose that F_i is a closed formula. If $T+A_{i-1}$ proves F_i, or $T+A_{i-1}$ proves $\neg F_i$, then we can ignore this formula, and set $A_i=A_{i-1}$. If T+A does not prove neither F_i, nor $\neg F_i$, then let us simply add F_i (or $\neg F_i$, if you like it better) to our set of new axioms, i.e., set $A_i = A_{i-1} \cup \{F_i\}$.

Etc., *ad infinitum*. As the result of this process we obtain a set of closed formulas $A = A_0 \cup A_1 \cup A_2 \cup ... \cup A_i \cup ...$.

Let us prove that T+A is a consistent complete theory.

Consistency. If T+A would be inconsistent, we would have a proof of [T+A]: $C \wedge \neg C$ for some formula C. If, in this proof, no axioms from the set A would be used, we would have a proof of [T]: $C \wedge \neg C$, i.e., T would be inconsistent.

Otherwise, the proof of [T+A]: $C \wedge \neg C$ contains a **finite number** of axioms B_1, ..., B_k from the set A. Let us arrange these axioms in the sequence, as we added them to the set A. Thus we have a proof of [T]: B_1, ..., $B_k \vdash C \wedge \neg C$. Then, by the N-elimination theorem, we have a proof of [T]: B_1, ..., $B_{k-1} \vdash \neg B_k$. But this is impossible − we added B_k to the set A just because $T+A_{i-1}$ could not prove neither B_k, nor $\neg B_k$. Q.E.D.

Completeness. We must verify that, for any closed formula F in the language of T, either $T+A \vdash F$, or $T+A \vdash \neg F$. Let us assume, this is not the case for some closed formula F. Of course, F appears in the above sequence F_0, F_1, F_2, ... as some F_i. If neither $T+A \vdash F$, nor $T+A \vdash \neg F$, then neither $T+A_{i-1} \vdash F_i$, nor $T+A_{i-1} \vdash \neg F_i$. In such a situation we would add F or ¬F to the set A, hence, we would have $T+A \vdash F$ or $T+A \vdash \neg F$. Q.E.D.

This completes the proof of Lindenbaum's Lemma.

Attention: non-constructive reasoning! T+A is a somewhat strange theory, because, in general, we do not have an algorithmic decision procedure for its set of axioms. Indeed, to decide, is some closed formula F an axiom of T+A, or not, we must identify F in the sequence F_0, F_1, F_2, ... as some F_i, and after

this, we must verify, whether T+A_{i-1} proves F_i, or T+A_{i-1} proves $\neg F_i$, or none of these. Thus, in general, T+A is not a formal theory in the sense of Section 1.1.

Proof of the Model Existence Theorem

(**Attention: non-constructive reasoning!**)

Inspired by the beautiful exposition in Mendelson [1997].

Step 1. We must build a model of T. What kind of "bricks" should we use for this "building"? **Idea #1**: let us use object constants of the language! So, in order to prepare enough "bricks", let us add to the language of T a countable set of new object constants d_1, d_2, d_3, ... (and extend the definitions of terms, atomic formulas and formulas accordingly, and add new instances of logical axioms accordingly). Let us prove that, if T is consistent, then this extended theory T_0 also is consistent.

If T_0 would be inconsistent, then, for some formula C, we could obtain a proof of [T_0]: $C \wedge \neg C$. If, in this proof, object constants from the set {d_1, d_2, d_3, ...} would not appear at all, then, in fact, we had a proof of [T]: $C \wedge \neg C$, i.e., we could conclude that T is inconsistent. But what, if some of the new object constants do appear in the proof of [T_0]: $C \wedge \neg C$? Then, let us replace these constants by any variables of T that do not appear in this proof (this is possible, since each predicate language contains a countable set of object variables). After these substitutions, the proof becomes a valid proof of T, because:

a) The logical axioms remain valid.

b) The non-logical axioms of T do not contain the object constants d_1, d_2, d_3, ..., so, they do not change.

c) Applications of inference rules MP and Gen remain valid.

Hence, [T]: $C' \wedge \neg C'$, where the formula C' has been obtained from C by the above substitutions. Thus, if T_0 would be inconsistent, then so would be T.

Step 2. The model we are building must contain all "objects" whose existence can be proved in T_0. **Idea #2:** for each formula F of T_0 having exactly one free variable (for example, x) let us add to the theory T_0 the axiom $\exists xF(x) \rightarrow F(d_i)$, where the constant d_i is unique for each F. If T_0 proves $\exists xF(x)$, then this d_i will represent in our model the object x having the property F. Let us prove that, if T is consistent, then this extended theory T_1 also is consistent. Note that

in T_1 the same language is used as in T_0.

To implement the Idea #2 correctly, first let us use the algorithm of the Exercise 4.3.3 printing out the sequence F_0, F_1, F_2, ... of all formulas in the language of T_0, and let us run through this sequence, processing only those formulas F_i that have exactly one free variable. Let us assign to each such formula F_i a unique constant $d_{c(i)}$ in such a way that $d_{c(i)}$ does not appear neither in the non-logical axioms of T, nor in F_i, nor in the axioms $\exists y F_j(y) \rightarrow F_j(d_{c(j)})$ for all formulas F_j preceding F_i in the sequence F_0, F_1, F_2, ... And, if x is the (only) free variable of F_i, let us adopt $\exists x F_i(x) \rightarrow F_i(d_{c(i)})$ as an axiom of T_1.

Now, let us assume that the extended theory T_1 is inconsistent, i.e., that, for some formula C in the language of T_0, we have a proof of $[T_1]$: $C \wedge \neg C$. In this proof, only a finite number n of axioms $\exists x F_i(x) \rightarrow F_i(d_{c(i)})$ could be used. Let us arrange these n axioms in order of increasing indices *i*, and let us denote this list by $A_1, A_2, ..., A_n$.

If n=0, then we have $[T_0]$: $C \wedge \neg C$, i.e., then T_0 is inconsistent. This is impossible.

If n>0, then let us consider A_n – the last axiom of the list: $\exists x F(x) \rightarrow F(d_{c(F)})$. And, in the proof of $[T_1]$: $C \wedge \neg C$, let us replace the constant $d_{c(F)}$ by some variable y that does not appear in this proof (this is possible, since each predicate language contains a countable set of variables). After this substitution, the proof remains a valid proof of T_1, because:

a) The logical axioms remain valid.

b) The non-logical axioms of T do not contain the constant c(F), they do not change.

c) The axiom $\exists x F(x) \rightarrow F(d_{c(F)})$ becomes $\exists x F(x) \rightarrow F(y)$. Since F does not contain the constant c(F), the premise $\exists x F(x)$ does not change.

d) The remaining n-1 axioms $\exists y F_j(y) \rightarrow F_j(d_{c(j)})$ of T_1 , i.e., the formulas of the list $A_1, A_2, ..., A_{n-1}$ do not contain the constant $d_{c(F)}$, they do not change.

e) Applications of inference rules MP and Gen remain valid.

Thus we have now a new proof of a contradiction:

$$[T_0]: A_1, A_2, ..., A_{n-1}, \exists x F(x) \rightarrow F(y) \vdash C' \wedge \neg C' ,$$

where the formula C' has been obtained from C by substituting y for $d_{c(F)}$. Then, by N-elimination theorem, there is a proof:

$$[T_0]: A_1, A_2, ..., A_{n-1} \vdash \neg(\exists xF(x) \rightarrow F(y)).$$

By Theorem 2.6.5, [L_1-L_{11}, MP]: $\neg(A \rightarrow B) \rightarrow A \wedge \neg B$. Thus, from $\neg(\exists xF(x) \rightarrow F(y))$ we can conclude $\exists x F(x) \wedge \neg F(y)$, and we have a proof of $\exists xF(x)$, and a proof of $\neg F(y)$. By applying Gen to the second formula, we obtain $\forall y \neg F(y)$, that is equivalent to $\neg \exists yF(y)$ (indeed, let us remind Section 3.2, Table 3.2, Group IV, constructively, $\forall x \neg B \leftrightarrow \neg \exists xB$). By Replacement Theorem 3, $\neg \exists yF(y)$ is equivalent to $\neg \exists xF(x)$. Thus, we have a proof of a contradiction $\exists x F(x) \wedge \neg \exists x F(x)$, where only the axioms [T_0]: A_1, A_2, ..., A_{n-1} are used.

Let us repeat the above chain of reasoning another $n-1$ times to eliminate **all** occurrences of the axioms $\exists xF_i(x) \rightarrow F(d_{c(i)})$ from our proof of a contradiction. In this way we obtain a proof of a contradiction in T_0. This is impossible. Hence, T_1 is a consistent theory.

Step 3. Idea #3: let us use the (non-constructive!) Lindenbaum's Lemma, and extend T_1 to a consistent complete theory T_2. Note that in T_2 the same language is used as in T_0.

Step 4. Let us define an interpretation M of the language of T_0, in which all theorems of T_2 will be true. Since all theorems of the initial theory T are theorems of T_2, this will complete our proof.

Idea #4: let us take as the domain D_M of the interpretation M the (countable! − verify!) set of all constant terms of T_0, i.e., terms that do not contain variables (this set of terms is not empty, it contains at least the countable set of object constants added in Step 1). And let us define interpretations of object constants, function constants and predicate constants as follows.

a) The interpretation of each object constant c is the constant c itself.

b) The interpretation of a function constant f is the "syntactic constructor function" f'', i.e., if f is an n-ary function constant, and t_1, ..., t_n are constant terms, then the value f''(t_1, ..., t_n) is defined simply as the character string "f(t_1, ..., t_n)" (quotation marks ignored).

c) The interpretation of a predicate constant p is the relation p'' such, if p is an n-ary predicate constant, and t_1, ..., t_n are constant terms, then p''(t_1, ..., t_n) is

defined as true in M if and only if T_2 proves $p(t_1, ..., t_n)$ (note that T_2 is a consistent complete theory, it proves either $p(t_1, ..., t_n)$, or $\neg p(t_1, ..., t_n)$, but not both!).

Step 5. To complete the proof, we must verify that, in the language of T_0, an arbitrary formula G is true in M if and only if T_2 proves G (let us denote this, as usual, by $T_2 \vdash G$). This will be done, if we will prove that, if $x_1, ..., x_m$ is any list of variables, containing all the free variables contained in the formula G, and $t_1, ..., t_m$ are constant terms, then

$$G(t_1, ..., t_m) \text{ is true in M if and only if } T_2 \vdash G(t_1, ..., t_m).$$

Note. Since T_2 is a *consistent and complete* theory, this is equivalent to

$$G(t_1, ..., t_m) \text{ is false in M if and only if } T_2 \vdash \neg G(t_1, ..., t_m).$$

Indeed, $G(t_1, ..., t_m)$ is a closed formula. If F is a closed formula, assume that F is true in M if and only if $T_2 \vdash F$. Then, a) if F is false, then $T_2 \vdash F$ is impossible, hence, $T_2 \vdash \neg F$; and b) if $T_2 \vdash \neg F$, then $T_2 \vdash F$ is impossible, hence, F cannot be true, and is false. Thus, F is false if and only if $T_2 \vdash \neg F$.

The proof will proceed by induction on the number of connectives and quantifiers in G.

Induction base: Here, G is an atomic formula $p(s_1, ..., s_n)$, where p is a predicate constant and $s_1, ..., s_n$ are terms. Then, $s_1, ..., s_n$ contain some of the variables $x_1, ..., x_m$. In $s_1, ..., s_n$, let us substitute for $x_1, ..., x_m$ the (constant) terms $t_1, ..., t_m$ respectively. In this way we obtain constant terms $s'_1, ..., s'_n$. Thus, $G(t_1, ..., t_m)$ is simply $p(s'_1, ..., s'_n)$. By definition (see Step 4), $p(s'_1, ..., s'_n)$ is true if and only if $T_2 \vdash p(s'_1, ..., s'_n)$, i.e., if and only if $T_2 \vdash G(t_1, ..., t_m)$. Q.E.D.

Induction step.

Let us consider a closed formula $G(t_1, ..., t_m)$. Let us denote $G(t_1, ..., t_m)$, $H(t_1, ..., t_m)$, $K(t_1, ..., t_m)$ simply by G, H, K correspondingly.

Case 1: G is $\neg H$. Then, H contains exactly the free variables of G, and we can consider $H(t_1, ..., t_m)$. According to the classical truth tables, G is true in M if and only if H is false in M. By the induction assumption and the above note, H is false in M if and only if $T_2 \vdash \neg H$, i.e., if and only if $T_2 \vdash G$. Q.E.D.

Case 2: G is $H \rightarrow K$. Then, H and K contain subsets of the free variables of G,

and we can consider $H(t_1, ..., t_m)$ and $K(t_1, ..., t_m)$. According to the classical truth tables, G is false in M if and only if H is **true** in M, and K is **false** in M. By the induction assumption and the above note, H is true in M if and only if $T_2 \vdash H$, and K is false in M if and only if $T_2 \vdash \neg K$. Hence,

$$\text{G is false in M if and only if } T_2 \vdash H \text{ and } T_2 \vdash \neg K.$$

Let us remind Theorem 2.2.1 and an equivalence from Section 2.6:

$$[L_1\text{-}L_{11}, \text{MP}]: \quad \neg(A \to B) \leftrightarrow A \wedge \neg B \quad .$$

In T_2, all the axioms of the classical logic are adopted, hence,

$$\text{G is false in M if and only if } T_2 \vdash \neg(H \to K),$$

$$\text{G is true in M if and only if } T_2 \vdash H \to K \text{ (by the above note),}$$

$$\text{G is true in M if and only if } T_2 \vdash G.$$

Q.E.D.

Case 3: G is $H \wedge K$. Then, H and K contain subsets of the free variables of G, and we can consider $H(t_1, ..., t_m)$ and $K(t_1, ..., t_m)$. According to the classical truth tables, G is true in M if and only if H is true in M, and K is true in M. By the induction assumption, H is true in M if and only if $T_2 \vdash H$, and K is true in M if and only if $T_2 \vdash K$. In T_2, all the axioms of the classical logic are adopted, hence, by Theorem 2.2.1,

$$T_2 \vdash H \text{ and } T_2 \vdash K \text{ if an only if } T_2 \vdash H \wedge K \ ,$$

$$\text{G is true in M if and only if } T_2 \vdash H \wedge K \ ,$$

$$\text{G is true in M if and only if } T_2 \vdash G.$$

Q.E.D.

Case 4: G is $H \vee K$. Then, H and K contain subsets of the free variables of G, and we can consider $H(t_1, ..., t_m)$ and $K(t_1, ..., t_m)$. According to the classical truth tables, G is true in M if and only if is true in M, or K is true in M. By the induction assumption and the above note, H is **false** in M if and only if $T_2 \vdash \neg H$, and K is **false** in M if and only if $T_2 \vdash \neg K$ (a somewhat smart idea to consider falsity instead of truth). Let us remind Theorem 2.2.1 and the Second de Morgan Law:

$$[L_1\text{-}L_9, \text{MP}]: \quad \neg(A \vee B) \leftrightarrow \neg A \wedge \neg B \quad .$$

In T_2, all the axioms of the classical logic are adopted, hence,

G is false in M if and only if $T_2 \vdash \neg(H \vee K)$,

G is true in M if and only if $T_2 \vdash H \vee K$ (by the above note),

G is true in M if and only if $T_2 \vdash$ G.

Q.E.D.

From now on, let us denote $G(t_1, ..., t_m)$, $H(x, t_1, ..., t_m)$ simply by G, H(x) correspondingly.

Case 5: G is ∃xH. Then, H contains all the free variables of G and, probably, x as a free variable, thus, we can consider H(x). Then, by definition, G is true in M if and only if H(x) is "true for some x", i.e., if and only if H(t) is true in M for some constant term t. By the induction assumption, H(t) is true in M if and only if $T_2 \vdash$ H(t).

If H does not contain x, then H(t) is true in M if and only if G is true in M. And, by Theorem 3.1.6,

$[L_1, L_2, L_{12}\text{-}L_{15}, MP, Gen]: G \leftrightarrow H$.

Q.E.D.

If H contains x as a free variable, let us remind our above Step 2. Since H(x) is a formula containing exactly one free variable, in T_2 we have the axiom $\exists xH(x) \rightarrow H(c_H)$, where c_H is an object constant.

First, let us assume that G is true in M. Then H(t) is true in M for some constant term t in M, hence, $T_2 \vdash$ H(t) for this particular t. Remind the axiom L_{13}: $F(t) \rightarrow \exists xF(x)$. Since t is a constant term, this axiom is valid for t. We need the following instance of L_{13}: $H(t) \rightarrow \exists xH(x)$. In T_2, all the axioms of the classical logic are adopted, hence, $T_2 \vdash H(t) \rightarrow \exists xH(x)$, and, by MP, $T_2 \vdash \exists xH(x)$, i.e., $T_2 \vdash$ G.

Now, let us assume that $T_2 \vdash$ G, i.e., $T_2 \vdash \exists xH(x)$. By the above-mentioned axiom, $T_2 \vdash \exists xH(x) \rightarrow H(c_H)$, where c_H is an object constant. Thus, $T_2 \vdash H(c_H)$. Since c_H is a constant term, by the induction assumption, if $T_2 \vdash H(c_H)$, then $H(c_H)$ is true in M. Hence, $H(c_H)$ is true in M, i.e., ∃xH(x) is true in M, and G is true in M. Q.E.D.

Case 6: G is ∀xH. Then, H contains all the free variables of G and, probably, x as a free variable, thus, we can consider H(x). Then, by definition, G is true in M if and only if H(x) is "true for all x", i.e., if and only if H(t) is true in M for all constant terms t. By the induction assumption, H(t) is true in M if and

only if $T_2 \vdash H(t)$.

Let us prove that

$$G \text{ is false in M if and only if } T_2 \vdash \exists x \neg H(x)$$

(a somewhat smart idea to consider falsity instead of truth).

First, let us assume that G is false in M. Then, by definition, H(t) is false in M for some constant term t. By the induction assumption, and by the above note, $T_2 \vdash \neg H(t)$. Let us remind the axiom L_{13}: $\neg H(t) \rightarrow \exists x \neg H(x)$. In T_2, all the axioms of the classical logic are adopted, hence, by MP, $T_2 \vdash \exists x \neg H(x)$.

Now, let us assume that $T_2 \vdash \exists x \neg H(x)$. Since H(x) is a formula containing exactly one free variable, in T_2 we have the axiom introduced in Step2: $\exists x \neg H(x) \rightarrow \neg H(c_{\neg H})$, where $c_{\neg H}$ is an object constant. Hence, by MP, $T_2 \vdash \neg H(c_{\neg H})$, i.e., T_2 does not prove $H(c_{\neg H})$. Then, by the induction assumption and the above note, $H(c_{\neg H})$ is false in M, i.e., $\forall x H(x)$ is false in M, i.e G is false in M.

Thus, we know that G is true in M if and only if T_2 does not prove $\exists x \neg H(x)$. Since T_2 is a complete theory, G is true in M if and only if $T_2 \vdash \neg \exists x \neg H(x)$. Now, let us remind from Section 3.2, Table 3.2, Group I, [L_1-L_{15}, MP, Gen]: $\neg \exists x \neg B \leftrightarrow \forall x B$. In T_2, all the axioms of the classical logic are adopted, hence, $T_2 \vdash \neg \exists x \neg H(x)$ if and only if $T_2 \vdash \forall x H(x)$, i.e., G is true in M if and only if $T_2 \vdash G$. Q.E.D.

This completes the proof of the Model Existence Theorem.

Attention: non-constructive reasoning! The above construction of the model M may seem "almost constructive". The domain D_M consists of all constant terms from the language of T_0. The axiom set of T_1 is algorithmically solvable (verify!). The interpretations of function constants are computable functions (verify!). But the interpretations of predicate constants? We interpreted each predicate constant p as the relation p" such that p"(t_1, ..., t_n) is true if and only if T_2 proves p(t_1, ..., t_n). This relation would be, in general, not algorithmically solvable, even if the axiom set of T_2 would be solvable! But, in general, the axiom set of theory T_2 (obtained by means of Lindenbaum's Lemma) is not algorithmically solvable! Thus, our construction of the model M is essentially non-constructive.

Exercise 4.3.4 (optional, course-work for smart students). Verify that the "degree of non-

constructivity" of the Model Existence Theorem is Δ_2^0 in the so-called arithmetical hierarchy. This became possible due to the improvements introduced by G. Hasenjäger. Hint: verify that all the predicates necessary for the proof are "computable in the limit". A function *p(x)* is called computable in the limit if and only if there is a computable function *g(x,n)* such that, for all x, $p(x)=\lim_{n\to\infty} g(x,n)$).

Exercise 4.3.5 (optional, course-work for smart students). If the language of first order theory T contains the equality predicate constant *x=y*, how this constant will be interpreted in the models, built for T according to the Model Existence Theorem? If the axioms of T imply the basic properties of equality (reflexivity, symmetry, transitivity and "indiscernibility of identicals") then only the following is guaranteed for the interpretation of *x=y*: it will always be interpreted as some equivalence relation over the domain of interpretation. In some contexts, the so-called **normal models** are preferable, where equality interpreted as **equality of domain elements**. Re-prove Model Existence Theorem by using normal models only. Or, see Mendelson [1997].

Consequences of Gödel's Completeness Theorem

From now on, in principle, we could forget our great ability of proving formulas in the classical predicate logic, that we developed in Section 3. Indeed, in order to verify, is a formula provable in [L_1-L_{15}, MP, Gen], or not, we can, instead of trying to build a proof from the axioms, try to verify, is this formula logically valid, or not. If it is, then, by Gödel's Completeness Theorem, it is provable in the classical logic, if not – it is not provable. For simple formulas, this method is really simpler than proving of formulas in [L_1-L_{15}, MP, Gen]. However, for the general case, this method does not work at all (see the Unsolvability Theorem below).

A second consequence was derived at the end of Section 4.1: **both our explications** of the assertion "G follows from A_1, ..., A_n" **are equivalent**, either as

[L_1-L_{15}, MP, Gen]: $A_1, \dots, A_n \vdash$ G, or as

"G is true under any interpretation, under which A_1, ..., A_n all are true".

The third consequence: Gödel's Completeness Theorem establishes a fundamental connection between provability/consistency and satisfiability.

Theorem 4.3.5. Consider a set of formulas A_1, ..., A_n in some predicate language.

a) It is **consistent** in the classical logic, if and only of it is **simultaneously satisfiable**;

b) It is **inconsistent** in the classical logic, if and only of it is **unsatisfiable simultaneously.**

c) For any closed formula G, [L_1-L_{15}, MP, Gen]: $A_1, \dots, A_n \vdash G$ if and only if the set $A_1, \dots, A_n, \neg G$ is unsatisfiable simultaneously.

Proof. Of course, (b) is a trivial reformulation of (a).

a) First, if a set of formulas is simultaneously satisfiable, i.e., its formulas are all true under some common interpretation J, then it is consistent according to Corollary 4.3.3(b).

On the other hand, if the set A_1, ..., A_n is consistent, then, according to the Model Existence Theorem, there is a model of A_1, ..., A_n, i.e., an interpretation making all these formulas true.

c) First, assume [L_1-L_{15}, MP, Gen]: $A_1, \dots, A_n \vdash G$ (Explication 1). This is equivalent to: G is true under any interpretation, under which A_1, ..., A_n are all true (Explication 2). Now, assume, there is an interpretation J making true all the formulas $A_1, \dots, A_n, \neg G$. Then, since A_1, ..., A_n are all true under J, so is G. But G is false under J. Contradiction, hence, the set $A_1, \dots, A_n, \neg G$ is unsatisfiable simultaneously.

On the other hand, assume, there is an interpretation J making true all the formulas $A_1, \dots, A_n, \neg G$. Then, A_1, ..., A_n are all true under J, but G is false under J. Hence (Explication 2), G does not follow from A_1, ..., A_n. This is equivalent to: in the classical logic, G cannot be proved from the hypotheses A_1, ..., A_n (Explication 1).

Q.E.D.

When trying to teach reasoning to computers, the conclusion of Theorem 4.3.5(c) is extremely important: one of the powerful proof procedures that can be implemented on computers, the so-called Method of analytic tableaux (Wikipedia), is based on the idea that in many practical situations, verifying of unsatisfiability is easier than proof searching. This method is explained in Section 6 below.

Extremely significant is also the fourth consequence of Gödel's Completeness Theorem: it shows that the "doubly non-constructive" notion of logically validity is at least 50% constructive – **semi-constructive**! Semi-solvable for computers!

Theorem 4.3.6 (Semi-solvability Theorem). There is an algorithm applicable to any predicate language and processing its formulas such that:

a) if the formula is logically valid, then the algorithm terminates and returns

“yes”;

b) if the formula is not logically valid, then the algorithm terminates and returns “no”, or it *does not terminate*.

Proof. According to the results of Exercises 1.1.4 and 1.1.7, the set of all formulas of the predicate language L provable in the classical logic, is algorithmically enumerable. Hence, by Gödel's Completeness Theorem, so is the set of all logically valid formulas in the language L: we can build an algorithm which, given the definition of the language L and working *ad infinitum*, prints out all the logically valid formulas of L (and only these formulas).

So, when processing a formula F, let us start this algorithm, and watch its output:

If we see the formula F printed, let us terminate and return “yes” (then F is, indeed, logically valid).

If we see the formula $\neg F$ printed, let us terminate and return “no” (then F is not logically valid, in fact, it is unsatisfiable).

If neither F, nor $\neg F$ will be printed at all, our processing will continue *ad infinitum* (in fact, such F is satisfiable, but not logically valid, but we may never become aware of it).

Q.E.D.

Computational complexity of the problem

Still, unfortunately, logical validity, being “50% constructive”, is not a 100% constructive notion. In 1936, Alonzo Church and Alan Turing proved that some predicate languages do not allow for an algorithm determining in all cases, is a given formula logically valid or not:

A. Church. A note on the Entscheidungsproblem. "Journal of Symb. Logic", 1936, vol.1, pp. 40-41.

A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. “Proceedings of the London Mathematical Society”, 2 (published 1937), 42 (1), pp. 230–265 (see also Turing’s proof in Wikipedia).

By applying a reduction theorem established by László Kalmár, this result can be greatly generalized:

L. Kalmar. Die Zurückführung des Entscheidungsproblems auf den Fall von Formeln mit einer einzigen, binären Funktionsvariablen. "Compositio Math.", 1937, Vol.4, pp.137-144.

Unsolvability Theorem. If a predicate language contains *at least one predicate constant that is at least binary*, then this language does not allow for an algorithm determining, is a given closed formula of this language **logically**

valid or not.

For details of the history, see Entscheidungsproblem in Wikipedia.

Thus, none of serious predicate languages allows for such an algorithm (the languages of first order arithmetic and set theory ZFC included). For details, see Mendelson [1997].

Of course, Gödel's Completeness Theorem implies the following equivalent

Unsolvability Theorem. If a predicate language contains at least one predicate constant that is at least binary, then this language does not allow for an algorithm determining, is a given closed formula **provable in the classical predicate logic, or not.**

Unsolvability Theorem and knowledge bases

If we have our knowledge base built by using some predicate language L, then, as noted above, we are interested in a **query processor** answering the questions:

"does formula G follow from the formulas A_1, ..., A_n?",

where A_1, ..., A_n (axioms) represent the knowledge stored in the knowledge base, and G is a query.

Thus, to build, for our knowledge base, a query processor, we must apply (or, invent) some algorithm allowing to determine (as fast as possible), given any formula G, does G follow from the axioms A_1, ..., A_n of the knowledge base, or not.

How **universal** could be made such a query processor? Could it be applicable:

a) only to our *specific* knowledge base A_1, ..., A_n, or

b) to *any knowledge base* that is using a specific predicate language L, or even,

c) to any knowledge base using *any predicate language*?

According to Lemma 4.1.3, G follows from A_1, ..., A_n if and only if the formula $A_1 \wedge ... \wedge A_n \rightarrow G$ is logically valid. Hence:

Semi-solvability Theorem for knowledge bases. There is a *universal algorithm* applicable to any predicate language L, to any knowledge bases A_1, ..., A_n and any queries G in L, such that:

a) if G follows from A_1, ..., A_n, then the algorithm terminates and returns "yes";

b) if $\neg G$ follows from A_1, ..., A_n, then the algorithm terminates and returns "no";

c) otherwise (i.e., neither G, nor $\neg G$ follow from A_1, ..., A_n) then the algorithm either terminates and returns "undecidable", or *does not terminate*.

Exercise 4.3.6. Verify this. (*Hint*: refine the above proof of the Semi-solvability Theorem.)

As we already know from the Exercise 1.1.6 (Section 1.1), if the set of axioms A_1, ..., A_n is **complete** (in the sense that for any closed formula G, either G, or $\neg G$ follows from the axioms), then the above situation (c) cannot occur:

Solvability Theorem for complete knowledge bases. If the set of axioms A_1, ..., A_n is **complete**, then the universal algorithm of the Semi-solvability Theorem terminates for any closed formula G giving a correct answer to the question "does G follow from A_1, ..., A_n".

Thus, the happy situation depends on specific features of the set A_1, ..., A_n. In general, determining, is our set of axioms complete, or not, usually, is a very hard task (remind our "language for people" and your attempts to propose complete system of axioms for it).

And, in general, the situation (c) cannot be excluded:

Unsolvability Theorem for knowledge bases. If a predicate language L contains at least one predicate constant that is at least binary, then L does not allow for an algorithm processing *any* knowledge bases A_1, ..., A_n, and *any* closed queries G in L, terminating in all cases and giving a correct answer to the question "does G follow from A_1, ..., A_n".

Exercise 4.3.7. Verify this.

Thus, a serious predicate language L does not allow for a **universal query processor**, applicable to *any* knowledge bases A_1, ..., A_n using the language L.

Note. We can try to improve the situation by exploring in parallel the queries G and $\neg G$. If our algorithm will answer "yes" for $\neg G$, that will mean the answer "no" for G (if our the axioms stored in our knowledge base are **consistent**). But, if the axioms are **incomplete**, i.e., they do not allow to decide between G and $\neg G$, then the process will not terminate, nevertheless.

This conclusion affects all the universal enough reasoning procedures for computers known today, such as Tableaux Method and Resolution Method considered in **Section 6** and **Section 7**.

Note. In principle, one can build even a *specific knowledge base* A_1, ..., A_n that does not allow for a query processor applicable to any closed queries G. As an example, we can take any finitely axiomatizable unsolvable mathematical theory, such as Von Neumann–Bernays–Gödel set theory (NBG). One can prove that, if NBG is consistent, then it is unsolvable: it does not allow for an algorithm, determining correctly in all cases, is a closed formula provable in NBG, or not. Thus, by taking the language of NBG and the (finite) set of axioms of NBG as A_1, ..., A_n, we obtain a single knowledge base that does not allow for a query processor applicable to any closed queries G.

However, the experience shows that the best universal algorithms (such as the above-mentioned Tableaux Method and Resolution Method) showing **unacceptable worst case** results (this is inevitable because of the Unsolvalility Theorem) perform **really good in many practical cases**.

Another approach allowing to build really usable knowledge bases: let us *restrict our predicate language* to make the problem of reasoning algorithmically solvable. For a successful attempt of this kind, see Description logic in Wikipedia.

Skolem's paradox

Initially, the Model Existence Theorem was proved in a weaker form in 1915 (by Leopold Löwenheim) and 1919 (by Thoralf Skolem): if a first order theory has a model, then it has a finite or countable model (**Löwenheim-Skolem Theorem**). Proof (possible after 1949): if T has a model, then T is consistent, hence, by Model Existence Theorem, T has a finite or countable model.

L. Löwenheim. Über Möglichkeiten im Relativkalkül. "Mathematische Annalen", 1915, Vol.76, pp. 447-470.

Th. Skolem. Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit und Beweisbarkeit mathematischen Sätze nebst einem Theoreme über dichte Mengen. *Videnskabsakademiet i Kristiania, Skrifter* I, No. 4, 1920, pp. 1-36.

Löwenheim-Skolem theorem (and the Model Existence Theorem) is steadily provoking the so-called **Skolem's Paradox**, first noted by Skolem in his address before the 5th Congress of Scandinavian Mathematicians (July 4-7, 1922):

Th. Skolem. Einige Bemerkungen zur axiomatischen Begründung der Mengenlehre. *Matematikerkongressen i Helsingfors den 4-7 Juli 1922, Den femte skandinaviska matematikerkongressen, Redogörelse*, Akademiska Bokhandeln, Helsinki, 1923, pp. 217-232.

Skolem called the effect "relativity of set-theoretic notions". Namely, in all formal set theories (for example, in ZFC) we can prove the existence of uncountable sets. Still, according to the Model Existence Theorem, if our

formal set theory is consistent, then there is a countable model in which all its axioms and theorems are true. Thus, a theory proves the existence of uncountable sets, yet it has a countable model! How could this be possible? Does it mean that all formal set theories are inconsistent?

In fact, Skolem's paradox is not a paradox at all. It should be rather called Skolem's effect − like as the photoelectric effect, it represents simply a **striking phenomenon**. Indeed, let J be a countable model of our formal set theory. In this theory, we can prove that the set r of all real numbers is uncountable:

$$\neg\exists f \text{ (f is 1-1 function from r into w)}, \qquad (1)$$

where w is the set of all natural numbers. What is the meaning of this theorem in the countable model J? Interpretations of r_J and w_J are subsets of the domain D_J, i.e., they both are countable sets, hence,

$$\exists f \text{ (f is 1-1 function from } r_J \text{ into } w_J). \qquad (2)$$

Interpretation of (1) in J is

$$\neg\exists f((\ f \in D_J\ ) \text{ and (f is 1-1 function from } r_J \text{ into } w_J)).$$

Hence, the mapping f of (2) does exist, yet it exists **outside the model** J! Do you think that f of (2) "must" be located in the model? Why? If you are living (as an "internal observer") within the model J, the set r_J seems uncountable to you (because you cannot find, in your world J, a 1-1 function from r_J into w_J). Still, for me (an "external observer") your uncountable r_J is countable − *in my world* I have a 1-1 function from r_J into w_J!

Hence, indeed, Skolem's Paradox represents simply a striking phenomenon. It is worth of knowing, yet there is no danger in it.

4.4. Constructive Propositional Logic – Kripke Semantics

Saul Aaron Kripke

S. Kripke (1965). Semantical analysis of intuitionistic logic. In: *J. N. Crossley, M. A. E. Dummet (eds.), Formal systems and recursive functions*. Amsterdam, North Holland, 1965, pp. 92-129.

Let us assume, again, that the formula F has been built of formulas B_1, B_2, ..., B_n ("atoms") by using propositional connectives only. According to the Completeness Theorem, F is provable in the **classical propositional logic** if

and only if it takes true values for any truth value assignments of B_1, B_2, ..., B_n. We know that many classically provable formulas cannot be proved in the constructive logic. Does that mean that constructive logic is “incomplete”? The so-called **Kripke semantics** shows a reasonable sense in which this logic can be regarded as complete.

Instead of simply computing truth values of F from truth values of B_1, B_2, ..., B_n, Kripke proposed to consider the **behavior** of F when the truth values of B_1, B_2, ..., B_n are changing gradually **from false to true** according to some "scenario".

Thus, Kripke proposed to replace the classical semantics (interpretation) of the propositional connectives (as defined by the classical truth tables) by a more complicated dynamic semantics.

Instead of simply saying that ¬F is true if and only if F is false, let us say that, ¬F is *true at some point in a scenario* if and only if, at this point, F is false and *remains false,* when the truth values of B_1, B_2, ..., B_n are changing according to the scenario.

Let o stand for implication, conjunction or disjunction. Instead of simply saying that FoG is true if and only if FoG is true according to the classical truth tables, let us say that, FoG is true at some point in a scenario if and only if, at this point, it is true and *remains true*, when the truth values of B_1, B_2, ..., B_n are changing according to the scenario.

Example 4.4.1. Let us consider the behavior of the classical axiom L_{11}: $B \vee \neg B$ in the scenario, where, at first, B is false, and at the next step it becomes true:

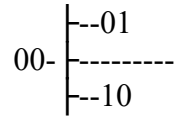
0 ---------- 1

How about ¬B? It becomes false at the next step, so, it cannot be qualified as true at the starting point, and must be qualified as false. Here we see the **main idea**: at the next step, ¬B becomes false, therefore, in a Kripke scenario, ¬B is qualified as false at the starting point as well. *For a formula to be qualified as true at some point, it must remain true at all the subsequent points.*

Since, at the starting point, B and ¬B both are qualified as false, then, at this point, the formula $B \vee \neg B$ must be qualified as false as well.

Thus, there is a simple Kripke scenario in which, at some point, $B \vee \neg B$ is false. Surprisingly, some time later (Lemma 4.4.3), we will derive from this simple fact that $B \vee \neg B$ cannot be proved in the constructive logic (we already know a much more complicated way of proving this fact from Section 2.8).

Example 4.4.2. Let us consider the behavior of that half of the First de Morgan Law: $\neg(A \wedge B) \rightarrow \neg A \vee \neg B$, that we failed to prove in the constructive logic. Let us consider a Kripke scenario, in which, at first, A and B both are false, and at the next step, two branches appear in the scenario: in the first branch: A remains false, and B becomes true, and in the second branch: A becomes true, and B remains false:



At the starting point: A is false, ¬A – also is false (for ¬A to be true, A must remain false at the next step, but in the second branch it doesn't). Similarly, at the starting point: B is false, ¬B – also false (for ¬B to be true, B must remain false at the next step, but in the first branch it doesn't). This means that, at the starting point, $\neg A \vee \neg B$ is false, but $\neg(A \wedge B)$ is true (because $A \wedge B$ is false, and it remains false in the both of branches), hence, at the starting point, $\neg(A \wedge B) \rightarrow \neg A \vee \neg B$ is false. Thus, there is a Kripke scenario in which, at some point, $\neg(A \wedge B) \rightarrow \neg A \vee \neg B$ is false. Surprisingly, some time later (Lemma 4.4.3), we will derive from this simple fact that the this half of the First de Morgan Law cannot be proved in the constructive logic. We failed to do this at all in Section 2.8!

Exercise 4.4.1. Investigate, in appropriate Kripke scenarios, the behavior of the following (only) classically provable formulas:

$$\neg\neg(A \vee B) \rightarrow \neg\neg A \vee \neg\neg B \quad ,$$
$$(A \rightarrow B) \rightarrow ((\neg A \rightarrow B) \rightarrow B) \quad ,$$
$$(A \rightarrow B) \vee (B \rightarrow A) \quad ,$$

and verify that, in some Kripke scenarios, these formulas are *not* true. Some time later (Lemma 4.4.3), we will derive from this simple fact that these formulas cannot be proved in the constructive logic. We failed to do this at all in Section 2.8! (Hint: consider the most simple scenarios first: 00--01, 00-10, 00-11, etc.)

More precisely, the definition of the **Kripke semantics** for the propositional language is as follows. Assume, the formula F has been built of the formulas B_1, B_2, ..., B_n ("atoms") by using propositional connectives only. Instead of simply considering truth values of F for all the possible assignments of truth values to B_1, B_2, ..., B_n, let us consider the behavior of F in all the possible Kripke scenarios, defined as follows.

Definition of Kripke scenarios. Each scenario s is a triple (b, ≤, t) of the following objects. First, b is a finite set of objects called nodes (or, states).

The second member ≤ is a partial ordering relationship between the nodes, i.e.,

for all $x, y, z \in b$: $x \le y \to (y \le z \to x \le z)$ (transitivity).

The third member t of the triple is a function (t means "true"). It associates with each node x a "growing" set t(x) of atoms, i.e., a subset of $\{B_1, B_2, ..., B_n\}$ in such a way that for all $x, y \in b$: $x \le y \to t(x) \subseteq t(y)$. If $B_i \in t(x)$, we say that B_i is true at the node x.

Note. In some other textbooks, Kripke scenarios are called **Kripke models**, or **Kripke structures**.

Thus, B_i is true at the node x if and only if $B_i \in t(x)$. We will denote this fact as x |= B_i ("at x, B_i is true", or "x forces B_i"). Since t is monotonic, if x |= B_i, then y |= B_i for all y after x, i.e., for all $y \in b$ such that $x \le y$. Thus, if B_i is true at some node x, then B_i remains true at all nodes after x.

Let us define the truth value of **x |= F** ("F is true at x", or "x forces F") for any formula F that has been built of the atoms $B_1, B_2, ..., B_n$ by using propositional connectives only.

1. Negation. Suppose, the truth value of x |= F is already defined for all $x \in b$. Then, x |= ¬F is defined to be **true** if and only if, for all $y \in b$ such that $x \le y$, y |= F is false (i.e., ¬(y |= F) is true according to the classical truth table of the negation connective). Else, x |= ¬F is defined to be false.

2. Implication, conjunction or disjunction. Suppose, the truth values of x |= F and x |= G are already defined for all $x \in b$. Then, x |= FoG is defined to be **true** if and only if, for all $y \in b$ such that $x \le y$, (y |= F)o(y |= G) is true according to the classical truth table of the connective o. Else, x |= FoG is defined to be false.

Lemma 4.4.1. For any formula F, any Kripke scenario (b, ≤, t), and any node $x \in b$: if x |= F, then y |= F for all $y \in b$ such that $x \le y$. Thus, if, in a Kripke scenario, a formula becomes true at some node, then it remains true at all the subsequent nodes. And, *if a formula is false at some node in a Kripke scenario, then it is false at the starting node of this scenario as well.*

Proof. By induction.

Induction base. See the definition above: if x |= B_i, then y |= B_i for all y after x, i.e., for all $y \in b$ such that $x \le y$.

Induction step.

1. Negation. Assume that x |= ¬F is true, i.e., that y |= F is false for all $y \in b$ such that $x \le y$. If $x \le y$, then is y |= ¬F true or false? By definition, y |= ¬F would be true if and only if z |= F would be false for all $z \in b$ such that

$y \leq z$. By transitivity of ≤, if $x \leq y$ and $y \leq z$, then $x \leq z$. By our assumption, if $x \leq z$, then z |= F is false. Hence, y |= ¬F is true. Q.E.D.

2. Implication, conjunction or disjunction. Assume, x |= FoG, i.e., according to the truth table of the connective o, (y |= F)o(y |= G) is true for all $y \in b$ such that $x \leq y$. If $x \leq y$, then is y |= FoG true or false? By definition, y |= FoG would be true if and only if (z |=F)o(z |= G) would be true for all $z \in b$ such that $y \leq z$. By transitivity of ≤, if $x \leq y$ and $y \leq z$, then $x \leq z$. By our assumption, if $x \leq z$, then (z |= F)o(z |= G) is true. Hence, y |= FoG. Q.E.D.

Exercise 4.4.2. Verify that if x is a *maximal* node in a scenario (b, ≤, t), then x |= F if and only if F is true at x according to the classical truth tables.

Kripke established that **a formula is provable in the constructive propositional logic if and only if it is true at all nodes in all Kripke scenarios.**

Theorem 4.4.2 (S. Kripke, completeness of the constructive propositional logic). A formula F is provable in the constructive propositional logic (i.e., [L_1-L_{10}, MP]: ⊢ F) if and only if F is true at the starting point of any Kripke scenario.

As usual, the hard part of the proof is establishing that "true is provable", i.e., if F is true at all nodes in all Kripke scenarios, then [L_1-L_{10}, MP]: ⊢ F (see Corollary 4.4.7 below). The easy part of the proof is, as usual, the soundness lemma:

Lemma 4.4.3. If [L_1-L_{10}, MP]: ⊢ F, then F is true at all nodes in all Kripke scenarios.

This lemma will follow from

Lemma 4.4.4. If F is any of the constructive axioms L_1-L_{10}, then, for any Kripke scenario (b, ≤, t), and any node $x \in b$: x |= F. Thus, the constructive axioms are true at all nodes in all Kripke scenarios.

and

Lemma 4.4.5. If, in a Kripke scenario (b, ≤, t), at the node $x \in b$: x |= F and x |= F→G, then x |= G. Hence, if F and F→G are true at all nodes in all Kripke scenarios, then so is G.

Proof of Lemma 4.4.3. Indeed, by Lemma 4.4.4, all the constructive axioms L_1-L_{10} are true at all nodes in all scenarios, and, by Lemma 4.4.5, the *Modus Ponens* rule preserves the property of being "true at all nodes in all scenarios". Q.E.D.

Note. Let us return to the above Example 4.4.2 and Exercise 4.4.1. We established that formulas

$$\neg(A \wedge B) \rightarrow \neg A \vee \neg B \quad ;$$
$$\neg\neg(A \vee B) \rightarrow \neg\neg A \vee \neg\neg B \quad ;$$
$$(A \rightarrow B) \rightarrow ((\neg A \rightarrow B) \rightarrow B)$$

are false at the starting nodes of some scenarios. Hence, by Lemma 4.4.3, these formulas cannot be proved in the constructive logic [L_1-L_{10}, MP]. We failed to prove this fact in Section 2.8!

Proof of Lemma 4.4.5. We know that x |= F→G means that (y |= F)→(y |= G) is true (according to the truth table of implication) for all $y \in b$ such that $x \leq y$. By Lemma 4.4.1, we know that y |= F for all $y \in b$ such that $x \leq y$. Hence, if y |= G would be false, then (y |= F)→(y |= G) also would be false. Hence, x |= G. Q.E.D.

Proof of Lemma 4.4.4.

L_1: B→(C→B)

x |= B→(C→B) is true if and only if (y |= B)→(y |= C→B) is true for all y≥x.

x |= B→(C→B) is false if and only if (y |= B)→(y |= C→B) is false for some y≥x.

How could (y |= B)→(y |= C→B) be false for some y≥x? According to the classical implication truth table, this could be only if and only if y |= B is true, and y |= C→B is false.

y |= C→B is true if and only if (z |= C)→(z |= B) is true for all z≥y.

y |= C→B is false if and only if (z |= C)→(z |= B) is false for some z≥y.

How could (z |= C)→(z |= B) be false for some z≥y? According to the classical implication truth table, this could be if and only if z |= C is true, and z |= B is false.

Summary:

x |= B→(C→B) is false
if and only if
∃y≥x (**y |= B is true** and y |= C→B is false)
if and only if
∃z≥y (z |= C is true and **z |= B is false**)

Hence, if x |= B→(C→B) is false, then there are y and z such that: x≤y≤z, **y |= B is true**, z |= C is true, and **z |= B is false**. By Lemma 4.4.1, if y≤z and y |= B is true, then z |= B is true. Contradiction with "z |= B is false". Thus, x |= B→(C→B) is true.

L_{10}: ¬B→(B→C)

x |= ¬B→(B→C) is false if and only if (y |= ¬B)→(y |= B→C) is false for some y≥x, i.e., if and only if y |= ¬B is true, and y |= B→C is false.

y |= ¬B is true if and only if z |=B is false for all z≥y.

y |= B→C is false if and only if (z |= B)→(z |= C) is false for some z≥y, i.e., if and only if z |= B is true, and z |= C is false.

Summary:

x |= ¬B→(B→C) is false
if and only if
∃y≥x (y |= ¬B is true and y |= B→C is false)
if and only if
∀z≥y (**z |=B is false**) ∃z≥y (**z |= B is true** and z |= C is false)

Hence, if x |= ¬B→(B→C) is false, then there is y≥x such that: a) ∀z≥y (**z |=B is false**), and b) ∃z≥y (**z |= B is true).** Contradiction. Thus, x |= ¬B→(B→C) is true.

L_3: $B \wedge C \rightarrow B$

x |= $B \wedge C \rightarrow B$ is false
if and only if
∃y≥x (y |= $B \wedge C$ is true and **y |= B is false**)
if and only if
∀z≥y **(z |=B is true** and z |= C is true)

Hence, there is y such that x≤y and y |= B is false. From ∀z≥y **(**z |=B is true) we obtain that y |= B is true. Contradiction. Thus, x |= $B \wedge C \rightarrow C$ is true.

L_4: $B \wedge C \rightarrow C$

Similarly.

L_5: $B \rightarrow (C \rightarrow B \wedge C)$

x |= $B \rightarrow (C \rightarrow B \wedge C)$ is false
if and only if
∃y≥x (**y |=B is true** and y |= $C \rightarrow B \wedge C$ is false)
if and only if
∃z≥y **(z |=C is true** and **z |=** $B \wedge C$ **is false**)

Hence, there are y, z such that x≤y≤z, y |= B is true, and z |= C is true, and **z |=** $B \wedge C$ **is false**. Then, by Lemma 4.4.1, u |= B is true, and u |= C is true, for all u≥z, i.e., **z |=** $B \wedge C$ **is true**. Contradiction. Thus, x |= $B \rightarrow (C \rightarrow B \wedge C)$ is true.

L_6: $B \to B \vee C$

$$x \models B \to B \vee C \text{ is false}$$
$$\text{if and only if}$$
$$\exists y \geq x\ (y \models \mathbf{B\ is\ true} \text{ and } y \models B \vee C \text{ is false})$$
$$\text{if and only if}$$
$$\exists z \geq y\ (z \models \mathbf{B\ is\ false} \text{ and } z \models C \text{ is false})$$

Hence, there are y, z such that $x \leq y \leq z$, $y \models B$ is true, and $z \models B$ is false. By Lemma 4.4.1, this is a contradiction. Thus, $x \models B \to B \vee C$ is true.

L_7: $C \to B \vee C$

Similarly.

L_8: $(B \to D) \to ((C \to D) \to (B \vee C \to D))$

$$x \models (B \to D) \to ((C \to D) \to (B \vee C \to D)) \text{ is false}$$
$$\text{if and only if}$$
$$\exists y \geq x\ (y \models \mathbf{B \to D\ is\ true} \text{ and } y \models (C \to D) \to (B \vee C \to D) \text{ is false})$$
$$\text{if and only if}$$
$$\exists z \geq y\ (z \models \mathbf{C \to D\ is\ true} \text{ and } z \models B \vee C \to D \text{ is false})$$
$$\text{if and only if}$$
$$\exists u \geq z\ (u \models B \vee C \text{ is true and } \mathbf{u \models D\ is\ false})$$

Hence, there are y, z, u such that $x \leq y \leq z \leq u$, $y \models B \to D$ is true, $z \models C \to D$ is true, and $u \models D$ is false. By Lemma 4.4.1, $u \models B \to D$ is true, and $u \models C \to D$ is true. Thus, if $u \models B$ would be true, then $u \models D$ also would be true. Hence, $u \models B$ is false. Similarly, $u \models C$ also is false. Hence, $u \models B \vee C$ is false. But we know that it is true. Contradiction. Thus, $x \models L_8$ is true.

L_2: (B→(C→D))→((B→C)→(B→D))

$$x \models (B \to (C \to D)) \to ((B \to C) \to (B \to D)) \text{ is false}$$
$$\text{if and only if}$$
$$\exists y \geq x\ (y \models B \to (C \to D) \text{ is true and } y \models (B \to C) \to (B \to D) \text{ is false})$$
$$\text{if and only if} \qquad \text{if and only if}$$
$$\forall z \geq y\ ((z \models B) \to (z \models C \to D)) \qquad \exists z \geq y\ (z \models B \to C \text{ is true and } z \models B \to D \text{ is false})$$
$$\text{if and only if} \qquad \text{if and only if}$$
$$\forall u \geq z\ ((u \models B) \to (u \models C)) \qquad \exists u \geq z\ (u \models B \text{ is true and } u \models D \text{ is false})$$

Hence, there are y, z, u such that $x \leq y \leq z \leq u$, $u \models B$ is true and $\mathbf{u \models D\ is\ false}$. From $\forall u \geq z\ ((u \models B) \to (u \models C))$ we obtain that $u \models C$ also is true, and from $\forall z \geq y\ ((z \models B) \to (z \models C \to D))$ – that $z \models C \to D$ is true. Then, by Lemma 4.4.1, $u \models C \to D$ also is true, i.e., $\forall v \geq u\ ((v \models C) \to (v \models D))$, in particular, $(u \models$

C)→(u |= D). Hence, **u|= D is true**. Contradiction. Thus, x |= L_2 is true.

L_9: (B→C)→((B→¬C)→¬B)

x |= (B→C)→((B→¬C)→¬B) is false
if and only if
∃y≥x (y |= B→C is true and y |= (B→¬C)→¬B is false)
if and only if — if and only if
∀z≥y ((z |= B)→(z |= C)) — ∃z≥y (z |= B→¬C is true and z |= ¬B is false)
if and only if — if and only if
∀u≥z ((u |= B)→(u |= ¬C)) — ∃u≥z (**u |= B is true**)

Hence, there are y, z, u such that x≤y≤z≤u , and **u |= B is true**. From **∀z≥y ((z |= B)→(z |= C))** we obtain that u |= C is true. From **∀u≥z ((u |= B)→(u |= ¬C))** we obtain that u |= ¬C is true, i.e., v |= C is false for some v≥u. By Lemma 4.4.1, if u |= C is true, then v |= C is true. Contradiction with "v |= C is false". Hence, x |= L_9 is true.

Exercise 4.4.3. Verify that, in the above recursive definition of x |= F, the item

2. Implication, conjunction or disjunction: x |= FoG is defined to be true if and only if, according to the truth table of the connective o, (y |= F)o(y |= G) is true for all $y \in b$ such that $x \le y$.

can be replaced by

2a. Implication ("non-monotonic" connective): x |= F→G is defined to be true if and only if, according to the truth table of implication, (y |= F)→(y |= G) is true for all $y \in b$ such that $x \le y$.

2b. Conjunction or disjunction ("monotonic" connectives): x |= FoG is defined to be true if and only if, according to the truth table of the connective o, (x |= F)o(x |= G) is true.

The hard part of the proof

Now, let us prove that, if F is true at all nodes in all Kripke scenarios, then F is provable in the constructive propositional logic. We will follow the paper

Judith L. Underwood. A Constructive Completeness Proof for Intuitionistic Propositional Calculus. TR-90-1179, December 1990, *Department of Computer Science, Cornell University*.

The smart idea is to **generalize the problem** in the following way. Instead of considering constructive provability of D_1, D_2, ..., D_m ⊢ $C_1 \vee C_2 \vee ... \vee C_n$ for arbitrary formulas D_1, D_2, ..., D_m, C_1, C_2, ..., C_n, i.e., let us consider ordered pairs of

sets ($\{D_1, D_2, ..., D_m\}$, $\{C_1, C_2, ..., C_n\}$). Let us call such pairs **sequents**. If S_1, S_2 are sets of formulas (S_1 may be empty), let us call the sequent (S_1, S_2) constructively provable if and only if [L_1-L_{10}, MP]: $S_1 \vdash VS_2$, where VS_2 denotes the disjunction of formulas contained in S_2. Moreover, let us consider **sets of sequents**. This will allow to carry out a specific induction argument (considering single formulas or single sequents does not allow such an argument!).

Let us say that a Kripke scenario (b, ≤, t) contains a **counterexample** for the sequent (S_1, S_2) if and only if the sequent is false at some node in the scenario (or, more precisely, if and only if there is $x \in b$ such that x |= F for all formulas $F \in S_1$ and not x |= G for all formulas $G \in S_2$).

Additionally, let us apply Corollary 8.1.2(b) of Theorem 8.1.1 to replace all negations ¬F by F→f, where f is an atomic formula, which is "always false", i.e., which, in a sequent (S_1, S_2), never belongs to S_1. Thus, formulas mentioned in the proof of the following Theorem 4.4.6 do not contain negations (but they may contain the specific atomic formula f).

Theorem 4.4.6. For any set S of sequents, either some sequent of S is constructively provable, or there is a Kripke scenario (b, ≤, t), which contains counterexamples for each sequent in S.

Proof. Let us start with a **proof overview**. We will consider the following cases:

Case 1. S contains (S_1, S_2) such that $A \wedge B \in S_1 \wedge \neg(A \in S_1 \wedge B \in S_1)$. Let us consider the set S' obtained from S by adding the "missing" formulas A, B to S_1, i.e., by replacing (S_1, S_2) by ($S_1 \cup \{A, B\}$, S_2). Let us verify that if Theorem is true for S', then it is true for S...

Case 2. S contains (S_1, S_2) such that $A \wedge B \in S_2 \wedge \neg(A \in S_2 \vee B \in S_2)$. Let us consider the following two sets: a) S' – obtained from S by adding the formula A to S_2, i.e., by replacing (S_1, S_2) by (S_1, $S_2 \cup \{A\}$). b) S" – obtained from S by adding the formula B to S_2, i.e., by replacing (S_1, S_2) by (S_1, $S_2 \cup \{B\}$). Let us verify that if Theorem is true for S' and S", then it is true for S...

Case 3. S contains (S_1, S_2) such that $A \vee B \in S_1 \wedge \neg(A \in S_1 \vee B \in S_1)$. Let us consider the following two sets: a) S' – obtained from S by adding the formula A to S_1, i.e., by replacing (S_1, S_2) by ($S_1 \cup \{A\}$, S_2). b) S" – obtained from S by adding the formula B to S_1, i.e., by replacing (S_1, S_2) by ($S_1 \cup \{B\}$, S_2). Let us verify that if Theorem is true for S' and S", then it is true for S...

Case 4. S contains (S_1, S_2) such that $A \vee B \in S_2 \wedge \neg(A \in S_2 \wedge B \in S_2)$. Let us consider the set S' obtained from S by adding the "missing" formulas A, B to S_2, i.e., by replacing (S_1, S_2) by (S_1, $S_2 \cup \{A, B\}$). Let us verify that if Theorem is true for S', then it is true for S...

Case 5. S contains (S_1, S_2) such that $A \rightarrow B \in S_1 \wedge \neg(A \in S_2 \vee B \in S_1)$. Let us consider the following two sets: a) S' – obtained from S by adding the formula A to S_2, i.e., by replacing (S_1, S_2) by (S_1, $S_2 \cup \{A\}$). b) S'' – obtained from S by adding the formula B to S_1, i.e., by replacing (S_1, S_2) by ($S_1 \cup \{B\}$, S_2). Let us verify that if Theorem is true for S' and S'', then it is true for S...

Case 6. S contains (S_1, S_2) such that $A \rightarrow B \in S_2$ and for every sequent $(T_1, T_2) \in S$, $\neg(S_1 \subseteq T_1 \wedge A \in T_1 \wedge B \in T_2)$. Let us consider the set S' obtained from S by adding the sequent ($S_1 \cup \{A\}$, B) to it. Let us verify that if Theorem is true for S', then it is true for S...

Case 7. None of the above cases hold for S. Then, Theorem is true for S – easy to verify...

The first six cases represent the induction argument: proving of Theorem for a sequent set S is reduced to proving it for some other sets – S' and S''. By iterating this reduction, we always arrive happily to the Case 7, where Theorem is easy to verify.

Indeed, let us denote by $universe(S_1, S_2)$ the set of all formulas and sub-formulas (of the formulas) contained in $S_1 \cup S_2$. Let us denote by $universe(S)$ the union of the universes of sequents from S.

Exercise 4.4.4. Verify that:

a) When, in the Cases 1-5, the sequent (S_1, S_2) is replaced by some other sequent (T_1, T_2), then

$$universe(T_1, T_2) \subseteq universe(S_1, S_2) \quad .$$

b) When, in the Case 6, because of the sequent (S_1, S_2), the sequent $(S_1 \cup \{A\}, B)$ is added to S, then

$$universe(S_1 \cup \{A\}, B) \subseteq universe(S_1, S_2) \quad .$$

c) For a given $universe(S)$, there exist no more than $N = 2^{|universe(S)|} + 1$ different sequents (S_1, S_2) such that $universe(S1, S2) \leq universe(S)$. And, no more than 2^N different sets of sequents.

Thus, any chain of iterated Cases 1-6 cannot be longer than 2^N+1 – either we will arrive at a set of sequents already built at a previous step, or we will arrive

at the Case 7.

Now – the proof as it should be.

Case 1. S contains (S_1, S_2) such that $A \wedge B \in S_1 \wedge \neg(A \in S_1 \wedge B \in S_1)$. Let us consider the set S' obtained from S by adding the "missing" formulas A, B to S_1, i.e., by replacing (S_1, S_2) by $(S_1 \cup \{A, B\}, S_2)$.

Let us verify that if Theorem is true for S', then it is true for S.

Assume, some sequent of S' is constructively provable, then it is $(S_1 \cup \{A, B\}, S_2)$ or some other sequent. If it is some other sequent, then it belongs to S, i.e., some sequent of S is constructively provable. If $(S_1 \cup \{A, B\}, S_2)$ is constructively provable, then so is (S_1, S_2). Indeed, if $S_1 \cup \{A, B\} \vdash VS_2$ is constructively provable, how to prove $S_1 \vdash VS_2$? Since S_1 contains $A \wedge B$, by axioms L_3 and L_3 we can derive A and B. After this, we can apply the proof of $S_1 \cup \{A, B\} \vdash VS2$. Hence, $S_1 \vdash VS_2$ is constructively provable.

On the other hand, if there is a Kripke scenario (b, ≤, t), which contains a counterexample for each sequent in S', then it contains also a counterexample for each sequent in S. Indeed, a sequent in S is either (S_1, S_2), or some other sequent. If it is some other sequent, then it belongs to S', i.e., (b, ≤, t) contains a counterexample for it. Does (b, ≤, t) contain a counterexample also for (S_1, S_2)? We know that it contains a counterexample for $(S_1 \cup \{A, B\}, S_2)$, i.e., for some $x \in b$, x |= F for all formulas $F \in S_1 \cup \{A, B\}$ and not x |= G for all formulas $G \in S_2$. Hence, (b, ≤, t) contains a counterexample also for (S_1, S_2). Q.E.D.

Case 2. S contains (S_1, S_2) such that $A \wedge B \in S_2 \wedge \neg(A \in S_2 \vee B \in S_2)$. Let us consider the following two sets:

a) S' – obtained from S by adding the formula A to S_2, i.e., by replacing (S_1, S_2) by $(S_1, S_2 \cup \{A\})$.

b) S'' – obtained from S by adding the formula B to S_2, i.e., by replacing (S_1, S_2) by $(S_1, S_2 \cup \{B\})$.

Let us verify that if Theorem is true for S' and S'', then it is true for S.

Assume, some sequent of S' and some sequent of S'' is constructively provable. The sequent of S' is $(S_1, S_2 \cup \{A\})$ or some other sequent. If it is some other sequent, then it belongs to S, i.e., some sequent of S is constructively provable. The sequent of S'' is $(S_1, S_2 \cup \{B\})$ or some other sequent. If it is some other

sequent, then it belongs to S, i.e., some sequent of S is constructively provable. So, let us consider the situation, when $(S_1, S_2 \cup \{A\})$ and $(S_1, S_2 \cup \{B\})$ both are constructively provable.

If $S_1 \vdash A \vee S_2$ and $S_1 \vdash B \vee S_2$ both are constructively provable, how to prove $S_1 \vdash VS_2$ (we know that S_2 contains $A \wedge B$)?

By Theorem 2.3.1, conjunction is distributive to disjunction:

$$[L_1\text{-}L_8, MP]: \ (A \wedge B) \vee C \leftrightarrow (A \vee C) \wedge (B \vee C) \ .$$

Hence, $[L_1\text{-}L_8, MP]: \ (A \vee S_2) \wedge (B \vee S_2) \rightarrow (A \wedge B) \vee S_2$. So, let us merge the proofs of $S_1 \vdash A \vee S_2$ and $S_1 \vdash B \vee S_2$, and let us append the proof of Theorem 2.3.1. Thus, we have obtained a proof of $S_1 \vdash (A \wedge B) \vee S_2$.

From Section 2.3 we know that in $[L_1\text{-}L_8, MP]$ disjunction is associative, commutative and idempotent. And, by Replacement Lemma 1(e):

$[L_1\text{-}L_8, MP]$ $A \leftrightarrow B \vdash A \vee C \leftrightarrow B \vee C$. Since S_2 contains $A \wedge B$, these facts allow, from a proof of $S_1 \vdash (A \wedge B) \vee S_2$, to derive a proof of $S_1 \vdash VS_2$.

On the other hand, if there is a Kripke scenario (b, ≤, t), which contains a counterexample for each sequent in S', then it contains also a counterexample for each sequent in S. Indeed, a sequent in S is either (S_1, S_2), or some other sequent. If it is some other sequent, then it belongs to S', i.e., (b, ≤, t) contains a counterexample for it. Does (b, ≤, t) contain a counterexample also for (S_1, S_2)? We know that it contains a counterexample for $(S_1, S_2 \cup \{A\})$, i.e., for some $x \in b$, x |= F for all formulas $F \in S_1$ and not x |= G for all formulas $G \in S_2 \cup \{A\}$. Hence, (b, ≤, t) contains a counterexample also for (S_1, S_2). Q.E.D.

If there is a Kripke scenario (b, ≤, t), which contains a counterexample for each sequent in S'', then it contains also a counterexample for each sequent in S. The argument is similar to the above.

Case 3. S contains (S_1, S_2) such that $A \vee B \in S_1 \wedge \neg(A \in S_1 \vee B \in S_1)$. Let us consider the following two sets:

a) S' – obtained from S by adding the formula A to S_1, i.e., by replacing (S_1, S_2) by $(S_1 \cup \{A\}, S_2)$.

b) S'' – obtained from S by adding the formula B to S_1, i.e., by replacing (S_1, S_2) by $(S_1 \cup \{B\}, S_2)$.

Let us verify that if Theorem is true for S' and S'', then it is true for S.

Assume, some sequent of S' and some sequent of S'' is constructively provable. The sequent of S' is $(S_1 \cup \{A\}, S_2)$ or some other sequent. If it is some other sequent, then it belongs to S, i.e., some sequent of S is constructively provable. The sequent of S'' is $(S_1 \cup \{B\}, S_2)$ or some other sequent. If it is some other sequent, then it belongs to S, i.e., some sequent of S is constructively provable. So, let us consider the situation, when $(S_1 \cup \{A\}, S_2)$ and $(S_1 \cup \{B\}, S_2)$ both are constructively provable.

Let us remind Exercise 2.3.2 [L_1, L_2, L_8, MP]: if $A_1, A_2, ..., A_n, B \vdash D$, and $A_1, A_2, ..., A_n, C \vdash D$, then $A_1, A_2, ..., A_n, B \vee C \vdash D$. Thus, if $S_1 \cup \{A\} \vdash VS_2$ and $S_1 \cup \{B\} \vdash VS_2$ both are constructively provable, then (since S_1 contains $A \vee B$) so is $S_1 U \{B\} \vdash VS_2$.

On the other hand, if there is a Kripke scenario (b, ≤, t), which contains a counterexample for each sequent in S', then it contains also a counterexample for each sequent in S. Indeed, a sequent in S is either (S_1, S_2), or some other sequent. If it is some other sequent, then it belongs to S', i.e., (b, ≤, t) contains a counterexample for it. Does (b, ≤, t) contain a counterexample also for (S_1, S_2)? We know that it is contains counterexample for $(S_1 \cup \{A\}, S_2)$, i.e., for some $x \in b$, x |= F for all formulas $F \in S_1 \cup \{A\}$ and not x |= G for all formulas $G \in S_2$. Hence, (b, ≤, t) contains a counterexample also for (S_1, S_2). Q.E.D.

If there is a Kripke scenario (b, ≤, t), which contains a counterexample for each sequent in S'', then it is also contains counterexample for each sequents in S. The argument is similar to the above.

Case 4. S contains (S_1, S_2) such that $A \vee B \in S_2 \wedge \neg(A \in S_2 \wedge B \in S_2)$. Let us consider the set S' obtained from S by adding the "missing" formulas A, B to S_2, i.e., by replacing (S_1, S_2) by $(S_1, S_2 \cup \{A, B\})$.

Let us verify that if Theorem is true for S', then it is true for S.

Assume, some sequent of S' is constructively provable, then it is $(S_1, S_2 \cup \{A, B\})$ or some other sequent. If it is some other sequent, then it belongs to S, i.e., some sequent of S is constructively provable. If $(S_1, S_2 \cup \{A, B\})$ is constructively provable, then so is (S_1, S_2). Indeed, if $S_1 \vdash (A \vee B) \vee S_2$ is constructively provable, how to prove $S_1 \vdash VS_2$ (where S_2 contains $A \vee B$)?

From Section 2.3 we know that in [L_1-L_8, MP] disjunction is associative,

commutative and idempotent. And, by Replacement Lemma 1(e):

[L_1-L_8, MP]: A↔B ⊢ $A \vee C \leftrightarrow B \vee C$. Since that S_2 contains AvB, these facts allow, from a proof of $S_1 \vdash (A \vee B) \vee S_2$, to derive a proof of $S_1 \vdash VS_2$.

On the other hand, if there is a Kripke scenario (b, ≤, t), which contains a counterexample for each sequent in S', then it contains also a counterexample for each sequent in S. Indeed, a sequent in S is either (S_1, S_2), or some other sequent. If it is some other sequent, then it belongs to S', i.e., (b, ≤, t) contains a counterexample for it. Does (b, ≤, t) contain a counterexample also for (S_1, S_2)? We know that it contains a counterexample for $(S_1, S_2 \cup \{A, B\})$, i.e., for some $x \in b$, x |= F for all formulas $F \in S_1$ and not x |= G for all formulas $G \in S_2 \cup \{A, B\}$. Hence, (b, ≤, t) contains a counterexample also for (S_1, S_2). Q.E.D.

If there is a Kripke scenario (b, ≤, t), which contains a counterexample for each sequent in S'', then it contains also a counterexample for each sequent in S. The argument is similar to the above.

Case 5. S contains (S_1, S_2) such that $A \to B \in S_1 \wedge \neg(A \in S_2 \vee B \in S_1)$. Let us consider the following two sets:

a) S' – obtained from S by adding the formula A to S_2, i.e., by replacing (S_1, S_2) by $(S_1, S_2 \cup \{A\})$.

b) S'' – obtained from S by adding the formula B to S_1, i.e., by replacing (S_1, S_2) by $(S_1 \cup \{B\}, S_2)$.

Let us verify that if Theorem is true for S' and S'', then it is true for S.

Assume, some sequent of S' and some sequent of S'' is constructively provable. The sequent of S' is $(S_1, S_2 \cup \{A\})$ or some other sequent. If it is some other sequent, then it belongs to S, i.e., some sequent of S is constructively provable. The sequent of S'' is $(S_1 \cup \{B\}, S_2)$ or some other sequent. If it is some other sequent, then it belongs to S, i.e., some sequent of S is constructively provable. So, let us consider the situation, when $(S_1, S_2 \cup \{A\})$ and $(S_1 \cup \{B\}, S_2)$ both are constructively provable.

We have two proofs: $S_1 \vdash A \vee S_2$ and S_1, B ⊢ VS_2, and we know that S_1 contains A→B. How to derive a proof of $S_1 \vdash VS_2$?

Since S_1 contains A→B, we have a proof of S_1, A ⊢ B. Together with S_1, B ⊢ VS_2 this yields a proof of S_1, A ⊢ VS_2. Of course, $VS_2 \vdash VS_2$. Now, let us remind Exercise 2.3.2 [L_1, L_2, L_8, MP]:

If A_1, A_2, ..., A_n, B ⊢ D, and A_1, A_2, ..., A_n, C ⊢ D, then A_1, A_2, ..., A_n, $B \vee C$ ⊢ D. Thus, S_1, $A \vee S_2$ ⊢ VS_2. Since we have a proof of S_1 ⊢ $A \vee S_2$, we have also a proof of S_1 ⊢ $A \vee S_2$.

On the other hand, if there is a Kripke scenario (b, ≤, t), which contains a counterexample for each sequent in S', then it contains also a counterexample for each sequent in S. Indeed, a sequent in S is either (S_1, S_2), or some other sequent. If it is some other sequent, then it belongs to S', i.e., (b, ≤, t) contains a counterexample for it. Does (b, ≤, t) contain a counterexample also for (S_1, S_2)? We know that it contains a counterexample for $(S_1, S_2 \cup \{A\})$, i.e., for some $x \in b$, x |= F for all formulas $F \in S_1$ and not x |= G for all formulas $G \in S_2 \cup \{A\}$. Hence, (b, ≤, t) contains a counterexample also for (S_1, S_2). Q.E.D.

If there is a Kripke scenario (b, ≤, t), which contains a counterexample for each sequent in S'', then it contains also a counterexample for each sequent in S. The argument is similar to the above.

Case 6. S contains (S_1, S_2) such that $A \rightarrow B \in S_2$ and for every sequent $(T_1, T_2) \in S$, $\neg(S_1 \subseteq T_1 \wedge A \in T_1 \wedge B \in T_2)$. Let us consider the set S' obtained from S by adding the sequent $(S1U\,A, B)$ to it.

Let us verify that if Theorem is true for S', then it is true for S.

Assume, some sequent of S' is constructively provable, then it is $(S_1 \cup \{A\}, B)$ or some other sequent. If it is some other sequent, then it belongs to S, i.e., some sequent of S is constructively provable. If $(S_1 \cup \{A\}, B)$ is constructively provable, then so is (S_1, S_2). Indeed, if S_1, A ⊢ B is constructively provable, then, by Deduction Theorem 1, S_1 ⊢ A→B, and S_1 ⊢ VS_2 (since S_2 contains A→B).

On the other hand, if there is a Kripke scenario (b, ≤, t), which contains a counterexample for each sequent in S', then, since S is a subset of S', this scenario contains also a counterexample for each sequent in S.

Case 7. None of the above cases hold for S. Hence, for every sequent $(S_1, S_2) \in S$ the following holds:

1) If $A \wedge B \in S_1$, then $A \in S_1 \wedge B \in S_1$,

2) If $A \wedge B \in S_2$, then $A \in S_2 \vee B \in S_2$,

3) If $A \vee B \in S_1$, then $A \in S_1 \vee B \in S_1$,

4) If $A \vee B \in S_2$, then $A \in S_2 \wedge B \in S_2$,

5) If $A \to B \in S_1$, then $A \in S_2 \vee B \in S_1$,

6) If $A \to B \in S_2$, then there is

$(T1, T2) \in S$ such that $S_1 \subseteq T_1 \wedge A \in T_2 \wedge B \in T_2$.

For this kind of sequent sets we have a very simple situation:

a) If, in some sequent $(S_1, S_2) \in S$ the sets S_1, S_2 contain the same formula A, then from L_6: $A \to A \vee B$ we can derive easily that [L_1-L_8, MP]: $S_1 \vdash VS_2$.

b) If the sets S_1, S_2 are disjoint for all sequents $(S_1, S_2) \in S$, then we must (and will) build a scenario, containing a counterexample for each sequent in S.

So, let us suppose that the sets S_1, S_2 are disjoint for all sequents $(S_1, S_2) \in S$, and let us define the following Kripke scenario (b, ≤, t):

b = S,

x≤y must be defined for every two members x, y of b, i.e., for every two sequents (S_1, S_2) and (T_1, T_2) in S. Let us define (S_1, S_2) ≤ (T_1, T_2) if and only if $S_1 \subseteq T_1$. Of course, $'\subseteq'$ is a partial ordering of b.

t must be a monotonic mapping from members of b to sets of atomic formulas. Let us define t(S_1, S_2) as the set of all atomic formulas in S_1. Of course, t is monotonic for $'\subseteq'$. (And, of course, f – our atomic "false", never belongs to t(S_1, S_2)).

Thus, (b, ≤, t) is a Kripke scenario. Let us prove that it contains a counterexample for each sequent in S. In fact, we will prove that for each sequent $(S_1, S_2) \in S$, and each formula F:

If $F \in S_1$, then (S_1, S_2) |= F.

If $F \in S_2$, then not (S_1, S_2) |= F.

This will mean that, (S_1, S_2) represents a counterexample for (S_1, S_2).

Of course, our proof will be by induction along the structure of the formula F.

a) F is an atomic formula.

If $F \in S_1$, then $F \in t(T_1, T_2)$ for every $(T_1, T_2) \in S$ such that (S_1, S_2)≤(T_1, T_2). Hence, (S_1, S_2) |= F.

If $F \in S_2$, then, since S_1 and S_2 are disjoint sets, $F \notin S_1$, and $F \notin t(S_1, S_2)$, i.e., not (S_1, S_2) |= F.

b) F is $A \wedge B$.

If $F \in S_1$, then, by (1), $A \in S_1 \wedge B \in S_1$. Hence, by induction assumption, $(S_1, S_2) \models A$ and $(S_1, S_2) \models B$, i.e., by Exercise 4.4.3, $(S_1, S_2) \models A \wedge B$.

If $F \in S_2$, then, by (2), $A \in S_2 \vee B \in S_2$. If $A \in S_2$, then, by induction assumption, not $(S_1, S_2) \models A$, i.e., by Exercise 4.4.3, not $(S_1, S_2) \models A \wedge B$. If $B \in S_2$ – the argument is similar.

c) F is $A \vee B$.

If $F \in S_1$, then, by (3), $A \in S_1 \vee B \in S_1$. If $A \in S_1$, then, by induction assumption, $(S_1, S_2) \models A$, i.e., by Exercise 4.4.3, $(S_1, S_2) \models A \vee B$. If $B \in S_1$ – the argument is similar.

If $F \in S_2$, then, by (4), $A \in S_2 \wedge B \in S_2$. By induction assumption, not $(S_1, S_2) \models A$ and not $(S_1, S_2) \models B$, i.e., by Exercise 4.4.3, not $(S_1, S_2) \models A \vee B$.

d) F is $A \rightarrow B$.

d1) $F \in S_1$. We must prove that $(S_1, S_2) \models A \rightarrow B$, i.e., that $(T_1, T_2) \models A \rightarrow B$ for each $(T_1, T_2) \in S$ such that $(S_1, S_2) \leq (T_1, T_2)$. So, let us assume that not $(T_1, T_2) \models A \rightarrow B$, i.e., that $(U_1, U_2) \models A$ and not $(U_1, U_2) \models B$ for some $(U_1, U_2) \in S$ such that $(T_1, T_2) \leq (U_1, U_2)$.

Since $A \rightarrow B \in S_1$, then also $A \rightarrow B \in U_1$, and, by (5), $A \in U_2 \vee B \in U_1$. By induction assumption, this means that not $(U_1, U_2) \models A$ or $(U_1, U_2) \models B$. Contradiction, hence, $(S_1, S_2) \models A \rightarrow B$.

d2) $F \in S_2$. We must prove that not $(S_1, S_2) \models A \rightarrow B$, i.e., that there is $(T_1, T_2) \in S$ such that $(S_1, S_2) \leq (T_1, T_2)$ and $(T_1, T_2) \models A$ and not $(T_1, T_2) \models B$.

Since $A \rightarrow B \in S_2$, by (6), there is $(T_1, T_2) \in S$ such that $(S_1, S_2) \leq (T_1, T_2)$ and $A \in T_1$ and $B \in T_2$. By induction assumption, this means that $(T_1, T_2) \models A$ and not $(T_1, T_2) \models B$. Q.E.D.

This completes the proof of Theorem 4.4.6.

Note. The above proof contains an **algorithm** allowing to find, for each set S of sequents, either a constructive proof of some sequent of S, or a Kripke scenario containing counterexamples for each sequent of S.

Corollary 4.4.7. If a formula F is true at the starting node of any Kripke scenario, then $[L_1\text{-}L_{10}, MP] \vdash F$ (i.e., F is provable in the constructive

propositional logic).

Indeed, let us consider the set of sequents {(0, {F})} consisting of a single sequent (0, {F}), where 0 is empty set. By Theorem 4.4.6, either the sequent (0, {F}) is constructively provable, or there is a Kripke scenario (b, ≤, t), which contains a counterexample for (0, {F}). Since F is true at all nodes in all Kripke scenarios, it cannot have counterexamples; hence, the sequent (0, {F}) (i.e., the formula F) is constructively provable.

Together with Lemma 4.4.3 this Corollary implies the above Theorem 4.4.2 – **Kripke's theorem on the completeness of the constructive propositional logic**: F is true at the starting point of any Kripke scenario if and only if F is provable in the constructive propositional logic.

It implies also

Corollary 4.4.8 (solvability of the constructive propositional logic). There is an **algorithm** allowing to determine for any propositional formula F, is this formula provable in the constructive propositional logic [L_1-L_{10}, MP], or not.

Thus, we have proved what Gerhard Gentzen established in 1934:

G. Gentzen. Untersuchungen über das logische Schliessen II. *Mathematische Zeitschrift*, 1934, Vol. 39, pp. 405-431.

The problem solved by this algorithm (determining the constructive provability of propositional formulas) belongs to the complexity class PSPACE-complete, as established by Richard Statman in 1979:

R. Statman. Intuitionistic propositional logic is polynomial-space complete, Theoretical Computer Science 9 (1979), pp. 67–72 (online copy available).

Corollary 4.4.9. If $F \vee G$ is true at the starting point of any Kripke scenario, then so is F, or so is G.

Proof. Assume, there is a scenario (b_1, $\leq_1$, t_1) such that x_1 |= F is false for some $x_1 \in b_1$, and a scenario (b_2, $\leq_2$, t_2) such that x_2 |= G is false for some $x_2 \in b_2$. We may assume that the (node) sets b_1 and b_2 do not intersect. Let us merge these scenarios by adding a new common starting node x_0, where all B_i are false. Then, x_0 |= F is false (Lemma 4.4.1), and x_0 |= G is false (similarly). Hence, according to the disjunction truth table, x_0 |= $F \vee G$ is false. But, x |= $F \vee G$ is true. Hence, x |= F is true, or x |= G is true. Q.E.D.

Now, we can prove a remarkable result established by Kurt Gödel in 1932:

K. Gödel. Zum intuitionistischen Aussagenkalkül. *Akademie der Wissenschaften in Wien, Mathematisch- naturwissenschaftliche Klasse, Anzeiger*, 1932, Vol.69, pp.65-66.

Theorem 4.4.10. If [L_1-L_{10}, MP]: $\vdash B \vee C$, then [L_1-L_{10}, MP]: $\vdash$ B or [L_1-L_{10}, MP]: $\vdash$ C. Thus, if the disjunction $B \vee C$ is constructively provable, then one of the formulas B, C also is constructively provable.

Proof. If [L_1-L_{10}, MP]: $\vdash B \vee C$, then, by Kripke's Completeness Theorem 4.4.2, $B \vee C$ is true at all nodes in all scenarios. Then, by Corollary 4.4.9, so is B or so is C. By Kripke's Completeness Theorem 4.4.2, this means that one of the formulas B, C is constructively provable. Q.E.D.

Let us remind the constructive interpretation of disjunction from Section 1.3:

- To prove $B \vee C$ constructively, you must prove B, or prove C. To prove $B \vee C$ classically, you may assume $\neg(B \vee C)$ as a hypothesis, and derive a contradiction. Having only such a "negative" proof, you may be unable to determine, which part of the disjunction $B \vee C$ is true – B, or C, or both.

Thus, according to Theorem 4.4.10, the constructive propositional logic [L_1-L_{10}, MP] supports the constructive interpretation of disjunction.

Exercise 4.4.5 (optional, for smart students). By adding the schema $(B \to C) \vee (C \to B)$ to the axioms of the constructive logic, we obtain the so-called *Gödel-Dummett logic* (Michael Dummett). Verify, that a propositional formula F is provable in Gödel-Dummett logic if and only if F is true at all nodes in all linear Kripke scenarios (i.e., in the scenarios that do not allow for branching). See also Intuitionistic Logic by Joan Moschovakis in Stanford Encyclopedia of Philosophy.

5. Normal Forms

Attention! The principal results of this Section are only valid for the **classical logic.**

The algorithms solving the problem of reasoning and implementable on computers, include, as the first step, transformation of formulas to more or less equivalent specific forms (the so-called **normal forms**) that are more convenient for efficient processing.

5.1. Negation Normal Form

See also Negation normal form in Wikipedia.

The first and simplest steps are elimination of equivalence and implication connectives.

Step 1: eliminate equivalence

We can eliminate all equivalence connectives, because $B \leftrightarrow C$ is only an abbreviation for $(B \rightarrow C) \wedge (C \rightarrow B)$. Why should we? Because, proving of B↔C consists of proving of B→C and proving (frequently, by a different method) of C→B. However, note that replacing of $B \leftrightarrow C$ by $(B \rightarrow C) \wedge (C \rightarrow B)$ (or, by two formulas: B→C and C→B) doubles the length of formulas to be processed.

Step 2: eliminate implication

After Step1, our formula will contain only implication, conjunction, disjunction and negation connectives. As the next step, we can eliminate implications.

Why should we eliminate implications? Because conjunction and disjunction are associative and commutative operations – very much like the addition and multiplication of numbers! See the example below.

The *classical logic* allows to do that. In Section 2.6 we proved that,

$$[L_1\text{-}L_{11}, MP]: \quad (A \rightarrow B) \leftrightarrow \neg A \vee B \quad .$$

By using this equivalence and replacement theorems, we can completely eliminate implication connectives, introducing negations and disjunctions

instead. For example, the formula $B \to (C \to D)$ can be transformed, first, into $\neg B \vee (C \to D)$, and then – into $\neg B \vee (\neg C \vee D)$. The latter formula is equivalent (in the classical logic only!) to the initial $B \to (C \to D)$.

Step 3: move negations down to atoms

Thus, after Step 2, our formula contains only conjunction, disjunction and negation connectives. Now, let us remind the two **de Morgan Laws**:

$$[L_1\text{-}L_{11}, MP]: \quad \neg(A \wedge B) \leftrightarrow \neg A \vee \neg B \quad ;$$
$$[L_1\text{-}L_9, MP]: \quad \neg(A \vee B) \leftrightarrow \neg A \wedge \neg B \quad ,$$

and the classical quantifier rules:

$$[L_1\text{-}L_{15}, MP, Gen]: \quad \neg \exists x F(x) \leftrightarrow \forall x \neg F(x) \quad ;$$
$$[L_1\text{-}L_{15}, MP, Gen]: \quad \neg \forall x F(x) \leftrightarrow \exists x \neg F(x) \quad .$$

By using these equivalencies, we can shift negations down – until the atoms of the formula. For example, let us transform the formula

$$((A \to B) \to C) \to B \wedge C \quad .$$

First, eliminate implications:

$\neg((A \to B) \to C) \vee (B \wedge C)$,
$\neg(\neg(A \to B) \vee C) \vee (B \wedge C)$,
$\neg(\neg(\neg A \vee B) \vee C) \vee (B \wedge C)$.

Apply de Morgan Laws:

$$(\neg\neg(\neg A \vee B) \wedge \neg C) \vee (B \wedge C) \quad ,$$
$$(\neg(\neg\neg A \wedge \neg B) \wedge \neg C) \vee (B \wedge C) \quad ,$$
$$((\neg\neg\neg A \vee \neg\neg B) \wedge \neg C) \vee (B \wedge C) \quad .$$

Now, let us remind the equivalence involving the classical **Double Negation Law**:

$$[L_1\text{-}L_{11}, MP]: \neg\neg A \leftrightarrow A.$$

It allows dropping of the excessive negations – we can replace $\neg\neg\neg A$ by $\neg A$ and $\neg\neg B$ – by B:

$$((\neg A \vee B) \wedge \neg C) \vee (B \wedge C) \quad .$$

This kind of formulas is called **negation normal forms (NNF)**. Namely, a formula is in a negation normal form, if it is built of atoms with or without negations by using conjunctions and disjunctions only. A formula in negation normal form contains only conjunctions, disjunctions and negations, and negations are located at the atoms only. Thus, we have obtained:

Theorem 5.1.1. In the classical logic, any formula of a predicate language can be transformed to an **equivalent** formula in a **negation normal form**. More

precisely, if F is a formula, then, following a simple algorithm, a formula F' can be constructed such that:

a) F' is in a negation normal form,

b) F' has the same free variables as F,

c) [L_1-L_{15}, MP, Gen]: F↔F'. If F does not contain quantifiers, then [L_1-L_{11}, MP, Gen]: F↔F'.

Proof. All the above-mentioned operations are replacing sub-formulas by equivalent formulas (in the classical logic). By Replacement Theorem 2, in this way, we obtain a formula that is equivalent to the initial formula. Q.E.D.

Negation normal forms represent the starting point for one of the efficient practical methods of reasoning that is implemented on computers – the so-called Method of Analytic Tableaux. This method is explained in Section 6.

Exercise 5.1.1. Transform to negation normal form the following formulas:

a) $((A \to B) \to A) \to A$ (Peirce's Law);

b) Axioms L_{14} and L_{15}.

Length of NNF

Exercise 5.1.2 (optional, for smart students). Verify that Step 2 and Step 3 increase the length of the formula *linearly*, and that these steps can be performed in *linear time* (see Time complexity in Wikipedia).

5.2. Conjunctive and Disjunctive Normal Forms

The next step is applicable to formulas or sub-formulas that do not contain quantifiers, i.e., to propositional formulas. After our formula is transformed into negation normal form, we can apply the following

Step 4: two algebras

After Step 3, our formula is built up by using:

a) atoms or atomic formulas

b) atoms or atomic formulas preceded by a negation,

c) conjunction and disjunction connectives.

Conjunction and disjunction are associative and commutative operations. By the behavior of "truth values", conjunction seems to be a kind of multiplication:

$$0\wedge 0=0, 0\wedge 1=1\wedge 0=0, 1\wedge 1=1 \quad ,$$

and disjunction – a kind of addition:

$$0\vee 0=0, 0\vee 1=1\vee 0=1, 1\vee 1=1 \quad .$$

However, for these operations **two** distributive laws are valid (Section 2.3) – conjunction is distributive to disjunction, and disjunction is distributive to conjunction:

[L_1-L_8, MP]: $(A\wedge B)\vee C\leftrightarrow(A\vee C)\wedge(B\vee C)$,

[L_1-L_8, MP]: $(A\vee B)\wedge C\leftrightarrow(A\wedge C)\vee(B\wedge C)$.

Thus, both of the two decisions are justified:

1) (Our first "algebra") Let us treat conjunction as multiplication and disjunction – as addition (+). Then the above formula

$((\neg A\vee B)\wedge\neg C)\vee(B\wedge C)$ takes the form ((A'+B)C')+BC (let us replace ¬A by the "more algebraic" A'). After this, the usual algebraic transformations yield the formula A'C'+BC'+BC. By the transformation back to logic we would obtain: $(\neg A\wedge\neg C)\vee(B\wedge\neg C)\vee(B\wedge C)$.

2) (Our second "algebra") Let us treat conjunction as addition (+) and disjunction – as multiplication. Then the above formula

$((\neg A\vee B)\wedge\neg C)\vee(B\wedge C)$ takes the form (A'B+C')(B+C). After this, the usual algebraic transformations yield the formula A'BB+A'BC+C'B+C'C. By the transformation back to logic we obtain:

$(\neg A\vee B\vee B)\wedge(\neg A\vee B\vee C)\wedge(\neg C\vee B)\wedge(\neg C\vee C)$.

However, additional non-numerical simplifying rules can be applied in these "algebras".

First rule – conjunction and disjunction are idempotent operations:

[L_1- L_5, MP]: $A\wedge A\leftrightarrow A$ (see Section 2.2).

[L_1, L_2, L_5, L_6-L_8, MP]: $A\vee A\leftrightarrow A$ (see Section 2.3).

Thus, in both of our "algebras": **A+A = AA = A**.

Second rule – $A\wedge\neg A$ (i.e., "false") is a kind of "zero" in the first "algebra", and a kind of "one" – in the second "algebra" (see Section 2.5):

[L_1-L_{10}, MP]: $B\vee(A\wedge\neg A)\leftrightarrow B$;

[L_1-L_{10}, MP]: $((A\wedge\neg A)\wedge B)\vee C\leftrightarrow C$.

Indeed, in the first "algebra", these formulas mean B+AA' = B and AA'B+C = C, i.e., we may think that AA'=0, B0=0, C+0=C. In the second "algebra", these formulas mean B(A+A') = B and (A+A'+B)C = C, i.e., we may think that

A+A'=1, B1=B, C+1=1.

Third rule – $A \vee \neg A$ (i.e., "true") is a kind of "one" in the first "algebra", and a kind of "zero" – in the second "algebra" (see Section 2.5):

[L_1-L_{11}, MP]: $B \wedge (A \vee \neg A) \leftrightarrow B$;

[L_1-L_{11}, MP]: $((A \vee \neg A) \vee B) \wedge C \leftrightarrow C$.

Indeed, in the first "algebra", these formulas mean B(A+A') = B and (A+A'+B)C = C, i.e., we may think that A+A'=1, B1=1, C+1=1. In the second "algebra". these formulas mean B+AA' = B and AA'B+C = C, i.e., we may think that AA'=0, B0=0, C+0=C.

Thus, **in both algebras,**

AA'=0, B0=0, C+0=C, A+A'=1, B1=B, C+1=1.

Let us continue our example:

1) (The first "algebra") The formula A'C'+BC'+BC is equivalent to A'C'+B(C'+C) = A'C'+B, or, if we return to logic: $(\neg A \wedge \neg C) \vee B$. Such disjunctions consisting of conjunctions are called **disjunctive normal forms** (DNFs). In a DNF, each conjunction contains each atom no more than once – either without negation, or with it. Indeed, if it contains some atom X twice, then: a) replace XX by X, or b) replace X'X' by X', or c) replace XX' by 0 (in the latter case – drop the entire conjunction from the expression).

In this way, for some formulas, we may obtain "zero", i.e., an **empty DNF**. Of course, such formulas take only false values ("false" is "zero" in the first "algebra"), i.e., are **unsatisfiable**.

And for some formulas, we may obtain "one", i.e., a kind of a **"full" DNF**. Such formulas take only true values ("true" is "one" in the first "algebra"), i.e., are **logically valid**.

2) (The second "algebra") The formula A'BB+A'BC+C'B+C'C is equivalent to A'B+A'BC+BC' = A'B(1+C)+BC' = A'B+BC', or, if we return to logic: $(\neg A \vee B) \wedge (B \vee \neg C)$. Such conjunctions consisting of disjunctions are called **conjunctive normal forms** (CNFs). In a CNF, each disjunction contains each atom no more than once – either without negation, or with it. Indeed, if it contains some atom X twice, then: a) replace XX by X, or b) replace X'X' by X', or c) replace XX' by 0 (in the latter case – drop the entire disjunction from the expression).

In this way, for some formulas, we may obtain "zero", i.e., an **empty CNF.** Of course, such formulas take only true values ("true" is "zero" in the second "algebra"), i.e., are **logically valid**.

And for some formulas, we may obtain "one", i.e., a kind of **"full" CNF**. Such

formulas take only false values ("false" is "one" in the second "algebra"), i.e., are **unsatisfiable**.

Thus, we have proved the following

Theorem 5.2.1. In the classical logic, every propositional formula can be transformed into an equivalent DNF and into an equivalent CNF. More precisely, assume, the formula F has been built of the formulas B_1, B_2, ..., B_n by using propositional connectives only. Then:

a) There is a formula F_1, which is in a (possibly empty or full) disjunctive normal form over B_1, B_2, ..., B_n such that [L_1-L_{11}, MP]: $F \leftrightarrow F_1$.

b) There is a formula F_2, which is in a (possibly empty or full) conjunctive normal form over B_1, B_2, ..., B_n such that [L_1-L_{11}, MP]: $F \leftrightarrow F_2$.

Proof. All the above-mentioned operations are replacing sub-formulas by equivalent formulas (in the classical logic). By Replacement Theorem 1, in this way, we obtain a formula that is equivalent to the initial formula. Q.E.D.

Exercise 5.2.1. a) Build DNFs and CNFs of the following formulas:

$$\neg(A \wedge B \rightarrow C) \ ,$$
$$(A \rightarrow B) \leftrightarrow (C \rightarrow D) \ ,$$
$$A \vee B \leftrightarrow C \vee D \ ,$$
$$A \wedge B \leftrightarrow C \wedge D \ .$$

b) Build DNFs and CNFs of the following formulas:

$$\neg(A \vee \neg A) \ ,$$
$$((A \rightarrow B) \rightarrow A) \rightarrow A \ ,$$
$$(A \rightarrow B) \rightarrow ((\neg A \rightarrow B) \rightarrow B) \ .$$

The notion of disjunctive normal form was known already in 1885 to Oscar Howard Mitchell:

O. H. Mitchell. On a New Algebra of Logic. In: *Studies in Logic by Members of the Johns Hopkins University,* 1885, pp. 72-106.

Length of CNF and DNF

Application of Step 4 may increase the length of the formula *exponentially* (see CNF example and DNF example in Wikipedia).

5.3. Prenex Normal Form

Let us consider an interpretation J of some predicate language L, such that the domain D_J contains an infinite set of objects. Under such interpretation, the "meaning" of formulas containing quantifiers may be more or less non-constructive, or, at least, "constructively difficult".

For example, the formula ∀xB(x) will be true, if B(x) will be true for all objects x in the (infinite!) set D_J. Thus, it is impossible to verify directly ("empirically"), is ∀xB(x) true or not. Saying that the formula ∀x∀y(x+y=y+x) is true under the standard interpretation of first order arithmetic, does not mean that we have verified this fact empirically – by checking x+y=y+x for all pairs of natural numbers x, y. Then, how do we know that ∀x∀y(x+y=y+x) is true? We could either postulate this feature of natural numbers directly (i.e., derive it inductively from partial "empirical evidence"), or prove it by using some set of axioms (i.e., derive it from other postulates). But, in general, formulas having the form ∀xB(x), are "constructively difficult".

The formula ∀x∃yC(x, y) may appear even more difficult: it will be true, if for each x in D_J we will be able to find y in D_J such that C(x, y) is true. Thus, thinking constructively, we could say that ∀x∃yC(x, y) is true, only, if there is an algorithm, which, for each x in D_J can find y in D_J such that C(x, y) is true.

For example, under the standard interpretation of first order arithmetic, the formula

$$\forall x \exists (x < y \wedge prime(y))$$

is true ("there are infinitely many prime numbers"). How do we know this? This “fact” was proved in 6th century BC. But a similarly quantified formula

,

represents the famous twin prime conjecture. Is it true or not? Until now, nobody knows the answer.

Exercise 5.3.1 (optional). Verify that the "meaning" of the formulas $\forall x \exists y \forall z D(x, y, z)$ and $\forall x \exists y \forall z \exists u F(x, y, z, u)$ may be even more non-constructive.

But how about the formula ∃xG(x)→∃yH(y)? Is it constructively more difficult than ∀x∃yC(x, y), or less? In general, we could prove that ∃xG(x)→∃yH(y) is true, if we had an algorithm, which, for each $x \in D_J$ such that G(x) is true, could find $y \in D_I$ such that H(y) is true, i.e., if ∀x∃y(G(x)→H(y)) would be true. We will establish below, that, in the classical logic, if G does not contain y, and H does not contain x, then the

formula $\exists xG(x)\rightarrow\exists yH(y)$ is equivalent to $\forall x\exists y(G(x)\rightarrow H(y))$. Thus, in general, the formula $\exists xG(x)\rightarrow\exists yH(y)$ is constructively as difficult as is the formula $\forall x\exists y(G(x)\rightarrow H(y))$.

To generalize this approach to comparing "constructive difficulty" of predicate formulas, the so-called **prenex normal forms** have been introduced:

a formula is in a prenex normal form if and only if it has all its quantifiers gathered in front of a formula that does not contain quantifiers. An example: $\forall x\exists y\forall z\exists u\,F(x,y,z,u)$, where the formula F does not contain quantifiers.

It appears, that in the classical logic, each formula can be transformed to an appropriate *equivalent* formula in a prenex normal form. To obtain this normal form, the following Lemmas 5.3.1–5.3.3 are used.

Lemma 5.3.1. If the formula G does not contain x as a free variable, then:

a) [L_1, L_2, L_5, L_{12}, L_{14}, MP, Gen]: $(G\rightarrow\forall xF(x))\leftrightarrow\forall x(G\rightarrow F(x))$.

b) [L_1, L_2, L_5, L_{12}-L_{15}, MP, Gen]: $(\exists xF(x)\rightarrow G)\leftrightarrow\forall x(F(x)\rightarrow G)$. What does it mean precisely?

c) [L_1-L_{11}, L_{12}-L_{15}, MP, Gen]: $(G\rightarrow\exists xF(x))\leftrightarrow\exists x(G\rightarrow F(x))$. More precisely:

[L_1-L_{11}, L_{12}-L_{15}, MP, Gen]: $(G\rightarrow\exists xF(x))\rightarrow\exists x(G\rightarrow F(x))$. This formula cannot be proved constructively! Explain, why. But the converse formula can be proved constructively:

[L_1, L_2, L_{13}-L_{15}, MP, Gen]: $\exists x(G\rightarrow F(x))\rightarrow(G\rightarrow\exists xF(x))$.

d) [L_1-L_{11}, L_{12}-L_{15}, MP, Gen]: $(\forall xF(x)\rightarrow G)\leftrightarrow\exists x(F(x)\rightarrow G)$. What does it mean precisely? More precisely:

[L_1-L_{11}, L_{12}-L_{15}, MP, Gen]: $(\forall xF(x)\rightarrow G)\rightarrow\exists x(F(x)\rightarrow G)$. This formula cannot be proved constructively! Explain, why. But the converse formula can be proved constructively:

[L_1, L_2, L_{13}-L_{15}, MP, Gen]: $\exists x(F(x)\rightarrow G)\rightarrow(\forall xF(x)\rightarrow G)$.

Proof. First, let us note that (a)← is an instance of the axiom L_{14}, and that (b)← is an instance of the axiom L_{15}.

Prove (a)→, (b)→, (c)←, (d)← as the Exercise 5.3.2 below.

It remains to prove (c)→ and (d)→ in the classical logic (constructive proofs are impossible here).

Let us prove (c)→: $(G\rightarrow\exists xF(x))\rightarrow\exists x(G\rightarrow F(x))$.

	First, let us prove:	$\neg\mathbf{G} \rightarrow ((G\rightarrow\exists xF(x))\rightarrow\exists x(G\rightarrow F(x)))$
(1)	$\neg G\rightarrow(G\rightarrow F(x))$	Axiom L_{10}.
(2)	$(G\rightarrow F(x))\rightarrow\exists x(G\rightarrow F(x))$	Axiom L_{13}: $F(x)\rightarrow\exists xF(x)$.
(3)	$\neg G\rightarrow\exists x(G\rightarrow F(x))$	From (1) and (2).
(4)	$\neg G \rightarrow ((G\rightarrow\exists xF(x)) \rightarrow \exists x(G\rightarrow F(x)))$	By L_1: $B\rightarrow(C\rightarrow B)$.
	Now, let us prove:	$\mathbf{G} \rightarrow ((G\rightarrow\exists xF(x))\rightarrow\exists x(G\rightarrow F(x)))$
(5)	G	Hypothesis assumed.
(6)	$G\rightarrow\exists xF(x)$	Hypothesis assumed.
(7)	$\exists xF(x)$	From (5) and (6).
(8)	$F(x)\rightarrow(G\rightarrow F(x))$	L_1: $B\rightarrow(C\rightarrow B)$.
(9)	$\forall x(F(x)\rightarrow(G\rightarrow F(x)))$	Gen.
(10)	$\exists xF(x)\rightarrow\exists x(G\rightarrow F(x))$	By [L_1, L_2, L_{12}-L_{15}, MP, Gen]: $\forall x(B\rightarrow C)\rightarrow(\exists xB\rightarrow\exists xC)$, Section 3.1.
(11)	$\exists x(G\rightarrow F(x))$	From (7) and (10).
(12)	$G \rightarrow ((G\rightarrow\exists xF(x))\rightarrow \exists x(G\rightarrow F(x)))$	By Deduction Theorem 2 (x is not a free variable in G and in $G\rightarrow\exists xF(x)$.
(13)	$\mathbf{G}\vee\neg\mathbf{G} \rightarrow ((G\rightarrow\exists xF(x))\rightarrow \exists x(G\rightarrow F(x)))$	From (4) and (12), by L_8. The total is [L_1, L_2, L_8, L_{10}, L_{12}-L_{15}, MP, Gen]
(14)	$(G\rightarrow\exists xF(x)\rightarrow \exists x(G\rightarrow F(x))$	$\mathbf{L_{11}}$: $G\vee\neg G$.

Finally, let us prove (d)→: $(\forall xF(x)\rightarrow G) \rightarrow \exists x(F(x)\rightarrow G)$. Let us denote this formula by H.

	First, let us prove:	$\forall\mathbf{xF(x)}\rightarrow H$
(1)	$\forall xF(x)$	Hypothesis assumed.
(2)	$\forall xF(x)\rightarrow G$	Hypothesis assumed.

(3)	G	From (1) and (2).
(4)	F(x)→G	By L_1: B→(C→B).
(5)	∃x(F(x)→G)	By L_{13}: F(x)→∃xF(x).
(6)	∀xF(x) →H	By Deduction Theorem 2.
	Now, let us prove:	**∃x¬F(x)**→H
(5)	¬F(x)	Hypothesis assumed.
(6)	¬F(x)→(F(x)→G)	L_{10}.
(7)	F(x)→G	From (5) and (6).
(8)	∃x(F(x)→G)	By L_{13}: F(x)→∃xF(x).
(9)	(∀xF(x)→G) → ∃x(F(x)→G)	By L_1: B→(C→B).
(10)	¬F(x)→H	By Deduction Theorem 2.
(11)	**∃x¬F(x)**→H	Gen and L_{15}: ∀x(¬F(x)→H)→ (∃x¬F(x)→H).
(12)	**¬∀xF(x)**→H	By Section 3.2, III-4. [L_1-L_{11}, L_{13}, L_{14}, MP, Gen]: ¬∀xF(x)→∃x¬F(x). **Axiom L_{11} is used here!**
(13)	**∀xF(x) v ¬∀xF(x)** → H	From (4) and (12), by L_8.
(14)	H	By L_{11}: **∀xF(x) v ¬∀xF(x)**

Q.E.D.

Exercise 5.3.2. a) Prove (a)→ of Lemma 5.3.1.

b) Prove (b)→ of Lemma 5.3.1.

c) Prove (c)← of Lemma 5.3.1. (Hint: prove (G→F(x))→(G→∃xF(x)), apply *Gen* and L_{15}.)

d) Prove (d)← of Lemma 5.3.1. (Hint: prove(F(x)→G)→(∀xF(x)→G), apply *Gen* and L_{15}.)

Lemma 5.3.2. If the formula G does not contain x as a free variable, then

a) [L_1-L_5, L_{13}-L_{15}, MP, Gen]: $\exists x\, F(x) \wedge G \leftrightarrow \exists x (F(x) \wedge G)$.

b) [L_1-L_5, L_{12}, L_{14}, MP, Gen]: $\forall x\, F(x) \wedge G \leftrightarrow \forall x (F(x) \wedge G)$.

c) [L_1-L_8, L_{13}-L_{15}, MP, Gen]: $\exists x\, F(x) \vee G \leftrightarrow \exists x (F(x) \vee G)$.

d) [L_1-L_8, L_{12}, L_{14}, MP, Gen]: $\forall x\, F(x) \vee G \leftrightarrow \forall x (F(x) \vee G)$.

Proof.

Prove (a, b, c, d) as the Exercise 5.3.3 below.

Q.E.D

Exercise 5.3.3. a) Prove (a, b) of Lemma 5.3.2. (Hint: to prove (a)→, first prove $F(x) \rightarrow (G \rightarrow \exists x (F(x) \wedge G))$, and apply L_{15}.)

b) Prove (c) of Lemma 5.3.2. (Hint: to prove (c)→, first prove $F(x) \rightarrow \exists x (F(x) \vee G)$, apply Gen and Axiom Axiom L_{15}; to prove (c)←, first prove $F(x) \vee G \rightarrow \exists x F(x) \vee G$, apply Theorem 2.3.1, if necessary.)

c) Prove (d) of Lemma 5.3.2. (Hint: to prove (d)→, apply D-elimination; to prove (d)←, apply L_{12} and D-elimination to obtain $\forall x (F(x) \vee G) \vdash \forall x\, F(x) \vee G$.)

Lemma 5.3.3. a) [L_1-L_{10}, L_{12}-L_{15}, MP, Gen]: ¬∃xF(x) ↔ ∀x¬F(x).

b) [L_1-L_{11}, L_{12}-L_{15}, MP, Gen]: ¬∀xF(x) ↔ ∃x¬F(x). More precisely:

[L_1-L_{11}, L_{13}, L_{14}, MP, Gen]: ¬∀xF(x) → ∃x¬F(x). This formula cannot be proved constructively! Why? But,

[L_1-L_{10}, L_{13}, L_{14}, MP, Gen]: ∃x¬F(x) → ¬∀xF(x).

Proof.

a) See Section 3.2, Group IV.

b)→. This is exactly Section 3.2, III-4.

b)←. See Section 3.2, Group III.

Q.E.D.

Let us remind that a formula is in a prenex normal form if and only if it has all its quantifiers gathered in front of a formula that does not contain quantifiers.

Theorem 5.3.4. In the classical logic, any formula of a predicate language can be transformed into an **equivalent** formula in a **prenex normal form**. More precisely, if F is a formula, then, following a simple algorithm, a formula F'

can be constructed such that:

a) F' is in a prenex normal form,

b) F' has the same free variables as F,

c) [L_1-L_{15}, MP, Gen]: F↔F'.

Proof. Let us start by an example:

$$\exists xG(x)\rightarrow\exists yH(y).$$

If H did not contain x as a free variable, then, by Lemma 5.3.1(b): $\exists xF(x)\rightarrow G \leftrightarrow \forall x(F(x)\rightarrow G)$, i.e., this formula would be equivalent to $\forall x(G(x)\rightarrow\exists yH(y))$. Now, let us consider the sub-formula $G(x)\rightarrow\exists yH(y)$. If G did not contain y as a free variable, then, by Lemma 5.3.1(c): $G\rightarrow\exists xF(x) \leftrightarrow \exists x(G\rightarrow F(x))$, the sub-formula would be equivalent to $\exists y(G(x)\rightarrow H(y))$. Hence, by Replacement Theorem 2, $\forall x(G(x)\rightarrow\exists yH(y))$ would be equivalent to $\forall x\exists y(G(x)\rightarrow H(y))$.

But, if H would contain x as a free variable, and/or G would contain y as a free variable? Then our "shifting quantifiers up" would be wrong – the formula $\forall x\exists y(G(x)\rightarrow H(y))$ would **not** be equivalent to $\exists xG(x)\rightarrow\exists yH(y)$.

To avoid this problem, let us use Replacement Theorem 3, which says that the meaning of a formula does not depend on the names of bound variables used in it. Thus, as the first step, in $\exists xG(x)$, let us replace x by another variable x_1 that does not occur neither in G, nor in H. Then, by Replacement Theorem 3, $\exists xG(x)$ is equivalent to $\exists x_1G(x_1)$, and by Replacement Theorem 2, $\exists xG(x)\rightarrow\exists yH(y)$ is equivalent to $\exists x_1G(x_1)\rightarrow\exists yH(y)$.

Now, $\forall x_1(G(x_1)\rightarrow\exists yH(y))$ is really equivalent to $\exists x_1G(x_1)\rightarrow\exists yH(y)$. As the next step, in $\exists yH(y)$, let us replace y by another variable y_1 that does not occur neither in G, nor in H. Then, by Replacement Theorem 3, $\exists yH(y)$ is equivalent to $\exists y_1H(y_1)$, and by Replacement Theorem 2, $G(x_1)\rightarrow\exists y_1H(y_1)$ is equivalent to $\exists y_1(G(x_1)\rightarrow H(y_1))$. And, finally, $\exists xG(x)\rightarrow\exists yH(y)$ is equivalent to $\forall x_1\exists y_1(G(x_1)\rightarrow H(y_1))$.

Now, we can start the general proof. In the formula F, let us find the **leftmost** quantifier having a propositional connective over it. If such a quantifier does not exist, the formula is in a prenex normal form. If such a quantifier exists, then F is in one of the following forms:

$$Q_qQ_q...Q_q(...(\neg QxG)...),\ \text{or}\ Q_qQ_q...Q_q(...(QxGooH)...),\ \text{or}\ Q_qQ_q...Q_q(...(GooQxH)...),$$

where $Q_qQ_q...Q_q$ are the quantifiers "already in the prefix", Q is the quantifier

in question, and oo is the propositional connective standing directly over Q.

In the first case, by Lemma 5.3.3, ¬QxG is equivalent to Q'x¬G, where Q' is the quantifier opposite to Q. By Replacement Theorem 2, $Q_qQ_q...Q_q$(...(¬QxG)...) is then equivalent to $Q_qQ_q...Q_q$(...(Q'x¬G)...), i.e., Q' has now one propositional connective less over it ((than had Q).

In the second case, as the first step, in QxG, let us replace x by another variable x_1 that does not occur in the entire formula F at all. Then, by Replacement Theorem 3, QxG is equivalent to Qx_1G_1, and by Replacement Theorem 2, $Q_qQ_q...Q_q$(...(QxGooH)...) is equivalent to $Q_qQ_q...Q_q$(...(Qx_1G_1ooH)...). Now, we can apply the appropriate case of Lemma 5.3.1 or Lemma 5.3.2, obtaining that Qx_1G_1ooH is equivalent to $Q'x_1$(G_1ooH), where Q' is the quantifier determined by the lemma applied. Then, by Replacement Theorem 2, $Q_qQ_q...Q_q$(...(Qx_1G_1ooH)...) is equivalent to $Q_qQ_q...Q_q$(...(Q'x_1(G_1ooH))...), i.e., Q' has now one propositional connective less over it (than had Q).

In the third case, the argument is similar.

By iterating this operation a finite number of times, we arrive at a formula F' which is in a prenex normal form, and which is (in the classical logic) equivalent to F. Q.E.D.

Note. In principle, the above-mentioned operations may be applied to the quantifiers of the formula in a different order. This is why many formulas admit several different prenex normal forms. For example, the above formula ∃xG(x)→∃xH(x) is equivalent not only to $\forall x_1\exists x_2(G(x_1)\rightarrow H(x_2))$, but also to $\exists x_2\forall x_1(G(x_1)\rightarrow H(x_2))$ (verify). This may come as a surprise, but it should not: $G(x_1)$ does not contain x_2, but $H(x_2)$ does not contain x_1.

As an example, let us obtain a prenex normal form of the following formula:

$$\forall x\,B(x)\vee\forall x\,C(x)\rightarrow\forall x\,D(x)\wedge(\neg\forall x\,F(x)) \quad .$$

First, assign unique names to bound variables:

$$\exists x_1\,B(x_1)\vee\forall x_2\,C(x_2)\rightarrow\forall x_3\,D(x_3)\wedge(\neg\forall x_4\,F(x_4)) \quad .$$

Process disjunction:

$$\exists x_1\forall x_2(B(x_1)\vee C(x_2))\rightarrow\forall x_3\,D(x_3)\wedge(\neg\forall x_4F(x_4)) \quad .$$

Process negation (∀ **changes to** ∃**!**):

$$\exists x_1\forall x_2(B(x_1)\vee C(x_2))\rightarrow\forall x_3\,D(x_3)\wedge\exists x_4\neg F(x_4) \quad .$$

Process conjunction:

$$\exists x_1 \forall x_2 (B(x_1) \vee C(x_2)) \to \forall x_3 \exists x_4 (D(x_3) \wedge \neg F(x_4)) \quad .$$

Process **implication** premise (**$\exists$ changes to $\forall$, $\forall$ changes to $\exists$**):

$$\forall x_1 \exists x_2 (B(x_1) \vee C(x_2) \to \forall x_3 \exists x_4 (D(x_3) \wedge \neg F(x_4))) \quad .$$

Process implication conclusion:

$$\forall x_1 \exists x_2 \forall x_3 \exists x_4 (B(x_1) \vee C(x_2) \to D(x_3) \wedge \neg F(x_4)) \quad .$$

The last two steps could be performed in the reverse order as well.

Attention! The step "assign unique names to bound variables" is essential! Without it, senseless results would be obtained. For example, converting of $\forall x G(x) \vee \forall x H(x)$ into $\forall x \forall x (G(x) \vee H(x))$ **is senseless**. In less emotional terms: formally, $\forall x \forall x (G(x) \vee H(x))$ is equivalent to the formula $\forall x (G(x) \vee H(x))$ that is not equivalent to $\forall x G(x) \vee \forall x H(x)$.

Exercise 5.3.4. Transform each of the following formulas into prenex normal forms. Write down every single step of the process. (Hint: the algorithm is explained in the proof of Theorem 5.3.4.)

a) $\exists x B(x) \to (\exists x C(x) \to \exists x D(x))$,

b) $\forall x \exists y B(x, y) \wedge \exists x C(x) \to \forall y \exists x D(x, y)$,

c) $\forall x B(x, y, z) \to \forall x C(x, y) \vee \exists y D(y, z)$,

d) $\forall x B(x) \to (\forall x C(x) \to (\forall x D(x) \to \forall x F(x)))$,

e) $((\exists x B(x) \to \exists x C(x)) \to \exists x D(x)) \to \exists x F(x)$.

Note. From a programmer's point of view, prenex normal forms are, in a sense, a crazy invention. In computer programming, you always try to reduce loop bodies, not to extend them as much as possible!

Exercise 5.3.5 (optional, for smart students). We may use transformation to prenex normal forms in proofs. More precisely, let us try extending the classical logic by introducing of the following additional inference rule (let us call it **PNF-rule**): given a formula F, replace it by some its prenex normal form F'. Verify, that, in fact, this rule does not extend the classical logic, i.e., if there is a proof of $F_1, F_2, ..., F_n \vdash G$ in [L_1-L_{15}, MP, Gen, PNF-rule], then there is a proof of the same in [L_1-L_{15}, MP, Gen]. (In some other texts, such rules are called **admissible rules**. Thus, the PNF-rule is an admissible rule in the classical logic.)

The notion of prenex normal forms and a version of Theorem 5.3.4 were known to Charles S. Peirce in 1885:

C. S. Peirce. On the algebra of logic: A contribution to the philosophy of notation. *American Journal of Mathematics*, 1885, vol.7, pp.180-202.

As noted by Alasdair Urquhart in the message [FOM] Prenex Normal Forms, Jul 14, 2007

(FOM is an automated e-mail list for discussing foundations of mathematics):

"On page 196 of that article, he [Peirce] gives a brief sketch of conversion to prenex normal form, remarking that it "can evidently be done."".

5.4. Skolem Normal Form

Skolem normal form was first introduced by Thoralf Skolem in 1928:

Th.Skolem. Über die mathematische Logik. "Norsk matematisk tidsskrift", 1928, vol.10, pp.125-142.

But the first very important idea was proposed by Skolem already in 1920:

Th. Skolem. Logisch-kombinatorische Untersuchungen über die Erfüllbarkeit und Beweisbarkeit mathematischen Sätze nebst einem Theoreme über dichte Mengen. *Videnskabsakademiet i Kristiania, Skrifter* I, No. 4, 1920, pp. 1-36.

In the previous sections, we considered only *equivalent* transformations: if the formula F is transformed into F', then [L_1-L_{15}, MP, Gen]: $F \leftrightarrow F'$. Skolem proposed to consider a wider class of transformations having the following property: if F is transformed into F', then the transformed formula F' is only "loosely equivalent" to the initial formula F. It appears that in this way we obtain the possibility of further "normalization" of formulas.

Let us remind that, in a predicate language L, a formula F is called *satisfiable* if and only if there is an interpretation of the language L under which F is true for some values of its free variables (if any). Skolem proposed to develop transformations having the following property: **the transformed formula F' is satisfiable if and only if so is the initial formula F**. Thus, two formulas F and F' are considered as "loosely equivalent", if we can prove that F is satisfiable, if and only so is F'.

Such transformations can be used in **refutation proofs**: to prove G, we assume $\neg G$ and try to derive a contradiction. Indeed, it follows from Gödel's Completeness Theorem that G is provable in the classical logic if and only if $\neg G$ is unsatisfiable. Thus, instead of trying to build a proof of a contradiction, we may try to detect, is $\neg G$ satisfiable or not. Hence, if we will transform $\neg G$ into a "better normalized" formula G' by retaining only satisfiability, then G' will be unsatisfiable if and only so is $\neg G$, i.e., if and only if G is provable.

Skolem's second idea: **introduction of new object constants and function constants allows to eliminate existential quantifiers**. This idea can be demonstrated on the following example: how could we simplify the formula $\forall x \exists y F(x, y)$? It asserts that for each x there is y such that F(x, y) is true. Thus, it asserts, that **there is a function** g, which selects for each value of

x a value of y such that F(x, y) is true. Thus, in a sense, $\forall x \exists y F(x, y)$ is "equivalent" to $\forall x F(x, g(x))$. In which sense? Exactly in the above-mentioned sense of "loose equivalence":

$\forall x \exists y F(x, y)$ is satisfiable if and only if $\forall x F(x, g(x))$ is satisfiable.

Indeed,

1. If $\forall x \exists y F(x, y)$ is satisfiable, then there is an interpretation J under which it is true, i.e., for each value of x there is a value of y such that F(x, y) is true. This allows us to define the following interpretation of the function constant g: g(x) is one of y-s such that F(x, y) is true under J. If we extend J by adding this interpretation of the function constant g, then we obtain an interpretation J' under which $\forall x F(x, g(x))$ is true, i.e., this formula is satisfiable.

2. If $\forall x F(x, g(x))$ is satisfiable, then there is an interpretation J under which it is true, i.e., for each value of x the formula F(x, g(x)) is true. Hence, in this interpretation, for each value of x there is a value of y (namely, g(x)) such that F(x, y) is true in J. Thus, $\forall x \exists y F(x, y)$ is true under J, i.e., this formula is satisfiable.

Note. In the first part of this proof, to define the function g, we need, in general, the Axiom of Choice. Indeed, if there is a non-empty set Y_x of y-s such that F(x, y) is true, to define g(x), we must choose a single element of Y_x. If we know nothing else about the interpretation J, we are forced to use the Axiom of Choice. But, if we know that the interpretation J has a countable domain, then we can define g(x) as the "least" y from the set Y_x. In this way we can avoid the Axiom of Choice.

The third idea is even simpler: the formula $\exists x F(x)$ asserts that there is x such that F(x) is true, so, let us denote by (an object constant) *c* one of these x-s, thus obtaining F(c) as a "normal form" of $\exists x F(x)$. Of course (verify),

$\exists x F(x)$ is satisfiable if and only if F(c) is satisfiable.

These ideas allow for transformation of any quantifier prefix $Qx_1...Qx_n$ into a sequence of universal quantifiers only:

Theorem 5.4.1 (Th. Skolem). Let L be a predicate language. There is an algorithm allowing to construct, for each *closed* formula F of this language, a *closed* formula F' (in a language L' obtained from L by adding a finite set of new object constants and new function constants – depending on F) such that:

a) F' is satisfiable if and only if F is satisfiable,

b) F' is in the form $\forall x_1 ... \forall x_n G$, where n≥0, and G does not contain quantifiers.

If a formula is in the form $\forall x_1 ... \forall x_n G$, where n≥0, and G does not

contain quantifiers, let us call it **Skolem normal form**. Thus, each closed formula can be transformed into a Skolem normal form in the following sense: for each closed formula F of a language L there is a Skolem normal form $|F|_{Sk}$ (in the language L extended by a finite set of Skolem constants and Skolem functions), which is satisfiable if and only if so is F.

Note. Computer science slang: the transformation procedure leading to Skolem normal forms is called "**skolemization**".

Note. Theorem 5.4.1 **does not** assert that a formula and its Skolem normal form are equivalent. It asserts only that the **satisfiability problem** of the first formula is equivalent to the satisfiability problem of the second formula. As already mentioned above, this is enough to allow using of skolemization in **refutation proofs**.

If we are interested in determining the satisfiability of formulas, then transformation to Skolem normal forms seems to be a promising method. Indeed, formulas $\forall x_1 ... \forall x_n G$ (where G does not contain quantifiers) are, perhaps, easier to analyze than formulas involving more complicated combinations of different quantifiers.

Proof of Theorem 5.4.1 First, let us obtain a prenex normal form F_1 of the formula F (see Section 5.3). Indeed, by Theorem 5.3.4, there is a simple algorithm, allowing to construct a closed formula F_1 such that F_1 is a prenex normal form, and, in the classical logic, $F \leftrightarrow F_1$. Of course, F_1 is satisfiable if and only if so is F.

If the quantifier prefix of F_1 starts with a sequence of existential quantifiers ($\exists\exists ... \exists\forall ...$), we will need the following lemma to eliminate these quantifiers:

Lemma 5.4.2. A closed formula $\exists x_1 ... \exists x_n H(x_1, ..., x_n)$ is satisfiable if and only if $H(c_1, ..., c_n)$ is satisfiable, where $c_1, ..., c_n$ are new object constants that do not occur in H.

After this operation, we have a closed prenex formula $H(c_1, ..., c_n)$ (in a language obtained from L by adding a finite set of new object constants, called **Skolem constants**), which is satisfiable if and only if so is F_1 (and F). The quantifier prefix of $H(c_1, ..., c_n)$ (if any) starts with a sequence of universal quantifiers ($\forall\forall ... \forall\exists ...$).

To proceed, we need the following

Lemma 5.4.3. A closed formula $\forall x_1 ... \forall x_n \exists y K(x_1, ..., x_n, y)$ is satisfiable if and only if $\forall x_1 ... \forall x_n \exists y K(x_1, ..., x_n, , g(x_1, ..., x_n))$ is satisfiable, where g is a new n-ary function constant (called **Skolem function**),

which does not occur in K.

By iterating these lemmas, we can transform the entire quantifier prefix of $H(c_1,\dots,c_n)$ to a sequence of universal quantifiers only ($\forall\forall\dots\forall$).

For example, the formula

$$\exists t\forall x\forall y\exists z\forall u\exists w F(t,x,y,z,u,w)$$

is satisfiable if and only if so is

$$\forall x\forall y\exists z\forall u\exists w F(c,x,y,z,u,w)$$

(where *c* is a constant that does not occur in F), and if and only if so is

$$\forall x\forall y\forall u\exists w F(c,x,y,g(x,y),u,w)\ ,$$

and finally if and only if so is the Skolem normal form:

$$\forall x\forall y\forall u F(c,x,y,g(x,y),u,h(x,y,u))\ ,$$

where g and h are functions that do not occur in F.

Exercise 5.4.1. a) Prove Lemma 5.4.2.

b) Prove Lemma 5.4.3.

How many new object constants and new function constants (Skolem constants and functions) do we need to obtain the final formula F'? The number of new symbols is determined by the number of existential quantifiers in the quantifier prefix of the prenex formula F_1. Indeed, a) the number of new object constants is determined by the number of existential quantifiers in front of the prefix, and b) the number of new function constants is determined by the number of existential quantifiers that follow after the universal ones.

This completes the proof of Theorem 5.4.1.

Exercise 5.4.2. Obtain Skolem normal forms of the formulas mentioned in Exercise 5.3.4.

See also:
"Skolemization" from The Wolfram Demonstrations Project. Contributed by Hector Zenil.

Now, let us obtain the **existential form** of Skolem's theorem.

Let F be a closed formula. F is logically valid if and only if $\neg F$ is unsatisfiable. Obtain a Skolem normal form of $\neg F$: $\forall x_1\dots\forall x_n G$ (where n≥0, and G does not contain quantifiers). Hence, by Theorem 5.4.1, F is logically valid if and only if $\forall x_1\dots\forall x_n G$ is unsatisfiable, hence, if and only if $\neg\forall x_1\dots\forall x_n G$ is logically valid. Thus, F is logically valid if and only so is $\exists x_1\dots\exists x_n\neg G$.

Thus we have proved the following

Theorem 5.4.4. Let L be a predicate language. There is an algorithm allowing to construct, for each closed formula F of this language, a closed formula F' (in a language L' obtained from L by adding a finite set of new object constants and new function constants – depending on F) such that:

a) F' is logically valid (or, provable in the classical logic) if and only if F is logically valid (or, provable in the classical logic),

b) F' is in form $\exists x_1 ... \exists x_n G$, where n≥0, and G does not contain quantifiers.

Exercise 5.4.3 (optional, for smart students). In his above-mentioned 1920 paper, for quantifier elimination, Skolem proposed **introduction of new predicate constants** (to the idea that function constants will do better, he arrived only in the 1928 paper). Do not read neither Skolem's papers, nor the above-mentioned online comments, and prove yourself that by introduction of new predicate constants, the satisfiability problem of any closed formula can be reduced to the satisfiability problem of a formula having the form $\forall x_1 ... \forall x_m \exists y_1 ... \exists y_n G$, where m, n≥0, and G does not contain quantifiers. Thus, function constants "will do better", indeed, see Theorem 5.4.1.

Exercise 5.4.4 (optional, compare with Exercise 5.3.5). Since, in general, Skolem normal form is not equivalent to the initial formula, we cannot use transformation to Skolem normal forms in the usual ("positive", or affirmative) proofs. But we may use it in "negative" (or, refutation) proofs, i.e., in proofs aimed at deriving a contradiction! More precisely, let us try extending the classical logic by introducing of the following additional inference rule (let us call it **SNF-rule**): given a formula F, replace it by some its Skolem normal form F' (such that the newly introduced object constants and function constants do not occur in the proof before F'). Verify, that, in fact, this rule does not extend the classical logic for refutation proofs, i.e., if, from a set of formulas F_1, F_2, ..., F_n, one can derive a contradiction by using [L_1-L_{15}, MP, Gen, SNF-rule], then one can do the same by using [L_1-L_{15}, MP, Gen]. (Thus, the SNF-rule is **admissible for refutation proofs** in the classical logic.)

5.5. Clause Form

Attention! The principal results of this Section are valid only for the **classical logic**!

Clause forms of propositional formulas

Which form is more "natural" – DNF, or CNF? Of course, CNF is more natural. Indeed, a DNF $D_1 \vee D_2 \vee ... \vee D_m$ asserts that one (or more) of the formulas D_i is true. This is a very complicated assertion – sometimes D_1 is true, sometimes D_2 is true, etc. But, if we have a CNF instead – $C_1 \wedge C_2 \wedge ... \wedge C_n$? It asserts that **all** the formulas C_i are true, i.e., we can

replace the long formula $C_1 \wedge C_2 \wedge ... \wedge C_n$ by a set of shorter formulas C_1, C_2, ..., C_n. For human reading and for computer processing, a set of shorter formulas is much more convenient than a single long formula.

Let us return to our example formula $((A \to B) \to C) \to B \wedge C$ of Section 5.2, for which we obtained a DNF $(\neg A \wedge \neg C) \vee B$ and a CNF:

$$(\neg A \vee B) \wedge (B \vee \neg C) \quad .$$

Without a transformation, DNF may be hard for reading and understanding. The CNF is more convenient – it says simply that $\neg A \vee B$ is true and $B \vee \neg C$ is true.

As another step, making the formulas easier to understand, we could apply the following equivalences:

[L_1-L_{11}, MP]: $\neg A \vee B \leftrightarrow A \to B$,

[L_1-L_{11}, MP]: $\neg A \vee \neg B \vee C \leftrightarrow A \wedge B \to C$,

[L_1-L_{11}, MP]: $\neg A \vee B \vee C \leftrightarrow A \to B \vee C$,

[L_1-L_{11}, MP]: $\neg A \vee \neg B \vee C \vee D \leftrightarrow A \wedge B \to C \vee D$,

etc.

Exercise 5.5.1. Verify these equivalences by proving that, generally (in the classical logic),

[L_1-L_{11}, MP]: $\neg A_1 \vee \neg A_2 \vee ... \vee \neg A_m \vee B_1 \vee B_2 \vee ... \vee B_n \leftrightarrow$

$$(A_1 \wedge A_2 \wedge ... \wedge A_m \to B_1 \vee B_2 \vee ... \vee B_n) \quad .$$

Thus, we can replace our set of two formulas $\neg A \vee B, B \vee \neg C$ by the set $A \to B, C \to B$. The conjunction of these two formulas is equivalent to the initial formula $((A \to B) \to C) \to B \wedge C$.

Formulas having the form

$$A_1 \wedge A_2 \wedge ... \wedge A_m \to B_1 \vee B_2 \vee ... \vee B_n \quad ,$$

or, equivalently,

$$\neg A_1 \vee \neg A_2 \vee ... \vee \neg A_m \vee B_1 \vee B_2 \vee ... \vee B_n \quad ,$$

where $A_1, A_2, ..., A_m, B_1, B_2, ..., B_n$ are atoms, are called **clauses**. Clauses are well suited for computer processing. Indeed, in the computer memory, we can represent the above formula simply as *a pair of sets of atoms* – the negative set $\{A_1, A_2, ..., A_m\}$ and the positive set $\{B_1, B_2, ..., B_n\}$.

What, if one (or both) of these sets is (are) empty?

If, in the formula $\neg A_1 \vee \neg A_2 \vee ... \vee \neg A_m \vee B_1 \vee B_2 \vee ... \vee B_n$, we have m = 0 and n > 0, then, of course, this formula asserts simply that $B_1 \vee B_2 \vee ... \vee B_n$, i.e., "converting" it into an implication with empty premise

$$\rightarrow B_1 \vee B_2 \vee ... \vee B_n$$

leads us to the following definition: the clause $\rightarrow B_1 \vee B_2 \vee ... \vee B_n$ means the same as $B_1 \vee B_2 \vee ... \vee B_n$.

If, in the formula $\neg A_1 \vee \neg A_2 \vee ... \vee \neg A_m \vee B_1 \vee B_2 \vee ... \vee B_n$, we have m > 0 and n = 0, then, of course, this formula asserts simply that $\neg A_1 \vee \neg A_2 \vee ... \vee \neg A_m$, i.e., "converting" it into an implication with empty consequence

$$A_1 \wedge A_2 \wedge ... \wedge A_m \rightarrow$$

leads us to the following definition: the clause $A_1 \wedge A_2 \wedge ... \wedge A_m \rightarrow$ means the same as $\neg(A_1 \wedge A_2 \wedge ... \wedge A_m)$.

If m=n=0, then, as an empty disjunction, the clause must be qualified as false.

Note. Clauses are similar to sequents – pairs of sets of formulas (S_1, S_2), used in the proof of Theorem 4.4.5 (completeness of the constructive propositional logic) in Section 4.4. In a sequent (S_1, S_2), the sets S_1, S_2 could contain arbitrary formulas, but, in a clause, S_1, S_2 are sets of atoms.

Sets (i.e., conjunctions) of clauses are called **clause forms** (in some texts – *clausal forms*). By Theorem 5.2.1, every propositional formula can be transformed to a (possibly empty, i.e., true) CNF. Since every conjunction member of a CNF represents, in fact, a clause, we have established the following

Theorem 5.5.1. In the classical logic, every propositional formula can be transformed to an equivalent clause form. More precisely, assume, the formula F is built of formulas B_1, ..., B_n by using propositional connectives only. Then there is a (possibly empty) clause form F' (i.e., a set of clauses) over B_1, ..., B_n such that [L_1-L_{11}, MP]: F ↔ conj(F'), where conj(F') denotes the conjunction of the clauses contained in the set F'.

For example, as we established above, the set $\neg A \vee B, B \vee \neg C$ (or, alternatively, $A \rightarrow B, C \rightarrow B$) is a clause form of the formula $((A \rightarrow B) \rightarrow C) \rightarrow B \wedge C$.

Exercise 5.5.2. Obtain clause forms of the formulas mentioned in the Exercise 5.2.1.

Clause forms (in a sense, “clouds of simple disjunctions”) are well suited for computer processing. In the computer memory, every clause

$$\neg A_1 \vee \neg A_2 \vee ... \vee \neg A_m \vee B_1 \vee B_2 \vee ... \vee B_n$$

can be represented as a pair of sets of atoms:

$$(-\{A_1, A_2, ..., A_m\}, +\{B_1, B_2, ..., B_n\}),$$

and every clause form – as a set of such pairs – i.e., it means less character string processing and less expression parsing.

Clause forms of predicate formulas

Of course (unfortunately), if we would insist that the clause form must be equivalent to the initial formula, then nothing comparable to clause forms could be obtained for predicate formulas. Still, transformation of predicate formulas to "clause forms" becomes possible, if we drop this requirement, and replace it by the requirement that the clause form must be simultaneously satisfiable if and only if the initial formula is satisfiable (or, equivalently, if proceeding, we wish only to retain the possibility of deriving contradictions). And – if we allow the Skolem style extending of the language by adding new object constants and new function constants.

Then, by Skolem's Theorem (Theorem 5.4.1), for each **closed** formula F, we can obtain a **Skolem normal form** $\forall x_1 ... \forall x_k G$, where k≥0, the formula G does not contain quantifiers, and this form is satisfiable if and only if so is F.

As the next step, let us **drop the universal quantifiers** in front of G. In this way the satisfiability is retained: G is satisfiable if and only if so is $\forall x_1 ... \forall x_k G$. (Remember that G, as a formula containing free variables, is considered as true in an interpretation if and only if it is true, in this interpretation, for all values of its free variables.)

Exercise 5.5.3. There is another way how to verify this. Use the *Gen*-rule and the axiom L_{12} to verify that G allows for deriving of contradictions if and only if $\forall x_1 ... \forall x_k G$ allows for it.

Thus, if we drop the universal quantifiers, then nothing is lost. The possibility of deriving contradictions is retained.

As the next step, by Theorem 5.2.1, let us convert G into a **CNF**, and then – into a clause form G', i.e., into a set of clauses G' = $\{C_1, C_2, ..., C_m\}$ (with **atomic** sub-formulas of G playing the role of atoms $B_1, B_2, ..., B_n$). Since conj(G') is equivalent to G, conj(G') is satisfiable if and only if so is F. Thus, the possibility of deriving contradictions is retained, again.

However, one more step is necessary to make processing of clauses easier: their “meanings” can be separated completely by renaming of variables in

such a way that **no two clauses contain common variables**.

Indeed, remember that we obtained the clauses from a closed formula in Skolem normal form: $\forall x_1 ... \forall x_k G$. Since conj(G') is equivalent to G,

$$\forall x_1 ... \forall x_k conj(G') \text{ is equivalent to } \forall x_1 ... \forall x_k G \ .$$

But $\forall x_1 ... \forall x_k conj(G')$ is equivalent to

$$(\forall x_1 ... \forall x_k C_1) \wedge ... \wedge (\forall x_1 ... \forall x_k C_m) \ .$$

According to the Replacement Theorem 3, we will obtain an equivalent formula, if we will rename the variables x_i in such a way that no two clauses will contain common variables.

Let us denote by G" = {C'$_1$, C'$_2$, ..., C'$_m$} the set of separated clauses obtained by renaming of variables. The set G" is called a *clause form* of the formula F.

Remember that **clauses** are disjunctions of **atomic** formulas (with, or without negation), i.e., disjunctions of formulas having the form $p(t_1, ..., t_m)$, or $\neg p(t_1, ..., t_m)$, where p is a predicate constant, and t$_1$, ..., t$_m$ are terms (containing variables and/or object constants).

In general, we are calling **clause form** any set of clauses, in which no two clauses contain common variables.

Thus, we have proved the following

Theorem 5.5.2. Let L be a predicate language. There is an algorithm allowing to construct, for each *closed* formula F of this language, a clause form, i.e., a finite set S of clauses (in a language L' obtained from L by adding a finite set of new object constants and new function constants – depending on F), such that no two clauses contain common variables, and the set S is simultaneously satisfiable if and only if F is satisfiable.

As an example, let us consider the formula asserting (in the language of first order arithmetic) that there are infinitely many prime numbers:

$$prime(x) \ : \ x>1 \wedge \neg \exists y \exists z (y>1 \wedge z>1 \wedge x=y*z) \ ,$$

$$\forall u \exists x (x>u \wedge prime(x)) \ ,$$

$$\forall u \exists x (x>u \wedge x>1 \wedge \neg \exists y \exists z (y>1 \wedge z>1 \wedge x=y*z)) \ . \qquad (1)$$

First, let us convert it into a prenex normal form:

$$\forall u \exists x \forall y \forall z (x>u \wedge x>1 \wedge \neg (y>1 \wedge z>1 \wedge x=y*z)) \ .$$

After this, let us eliminate ∃x by introducing a Skolem function x=g(u):

$$\forall u \forall y \forall z (g(u)>u \wedge g(u)>1 \wedge \neg (y>1 \wedge z>1 \wedge g(u)=y*z)) \ .$$

This is a Skolem normal form, so, let us drop the universal quantifiers:

$$g(u)>u \wedge g(u)>1 \wedge \neg(y>1 \wedge z>1 \wedge g(u)=y*z) \quad ,$$

and let us convert this quantifier-free formula into a conjunctive normal form:

$$g(u)>u \wedge g(u)>1 \wedge (\neg(y>1) \vee \neg(z>1) \vee \neg(g(u)=y*z)) \quad .$$

By dropping the conjunctions we obtain 3 clauses:

$$g(u)>u \quad , \quad g(u)>1 \quad , \quad \neg(y>1) \vee \neg(z>1) \vee \neg(g(u)=y*z) \quad .$$

As the last step, we must rename the variables in such a way that no two clauses contain common variables:

$$\begin{gathered} g(u_1)>u_1 \quad , \\ g(u_2)>1 \quad , \\ \neg(y>1) \vee \neg(z>1) \vee \neg(g(u_3)=y*z) \quad . \end{gathered}$$

or, alternatively,

$$\begin{gathered} \rightarrow g(u_1)>u_1 \quad , \\ \rightarrow g(u_2)>1 \quad , \\ y>1, z>1, g(u_3)=y*z \rightarrow \quad . \end{gathered}$$

These sets of 3 formulas are clause forms of the formula (1). *They are simultaneously satisfiable if and only if satisfiable is the initial formula (1).*

By renaming of variables, the "meanings" of clauses become completely separated. For example, the clause $g(u_1)>u_1$ means $\forall u_1 g(u_1)>u_1$, independently of what is meant in other clauses.

Exercise 5.5.4. Obtain clause forms of the formulas mentioned in the Exercise 5.3.4 (assume that B, C, D, F are predicate constants).

Clause form of a set of formulas

Knowledge bases are, as a rule, large sets of closed formulas $F_1, F_2, \ldots, F_n$, i.e., theoretically, large conjunctions $F_1 \wedge F_2 \wedge \ldots \wedge F_n$ of closed formulas. The clause form of this set can be obtained, simply as the **union** of clause forms of separate formulas F_i. However, each formula must be "kept separated" during the entire process:

a) when transforming to Skolem normal forms, the name of each new Skolem constant and Skolem function must be chosen as "completely new" with respect to the entire process;

b) when renaming clause variables, at the end, we must guarantee that no two clauses of the entire process contain common variables.

If these conditions are respected, then a clause form of the set $F_1, F_2, \ldots, F_n$ is simultaneously satisfiable if and only if so is the set itself:

Theorem 5.5.3. Let L be a predicate language. There is an algorithm allowing to construct, for each finite set of *closed* formulas $\boldsymbol{F}=\{F_1,\ldots,F_n\}$ of this language, a clause form, i.e., a finite set **S** of clauses (in a language L' obtained from L by adding a finite set of new object constants and new function constants – depending on $\boldsymbol{F}$), such that no two clauses contain common variables, and the set **S** is simultaneously satisfiable if and only if so is $\boldsymbol{F}$.

6. Tableaux Method

Attention! The principal results of this section are valid only for the **classical logic**!

In Section 6 and Section 7, we consider two practical methods for solving the **problem of reasoning** for the classical logic (in other words: how to build a query processor for knowledge bases that are using predicate languages and the classical logic).

Let us remind the problem: given a predicate language L, we are interested in methods allowing to answer the following questions:

"does some formula G (a query) follow from the formulas $A_1, \dots, A_n$
(the axioms, stored in a knowledge base)?"

In Section 1.3 and Section 4.1 we introduced two different explications of these informal questions:

1) Can the formula F be derived from the hypotheses $A_1, \dots, A_n$ by using the classical predicate logic? In other words, does there exist a proof

$$[L_1\text{-}L_{15}, \text{MP, Gen}]: \quad A_1, \dots, A_n \vdash G?$$

2) Is the formula G true in every interpretation of the language L, in which are true all the formulas $A_1, \dots, A_n$?

It follows from Gödel's Completeness Theorem, that these two explications are equivalent (see Section 4.3). Thus, we can speak simply about "G following from $A_1, \dots, A_n$ ", and – depending on the situation – switch to the more convenient of the explications.

However, a serious obstacle is posed by the Unsolvability Theorem: no algorithm will be able solve all cases of the reasoning problem correctly, i.e., in all cases answer the above questions correctly, "yes" or "no". On the other hand, the Semi-solvability Theorem guarantees at least the possibility of algorithms giving correctly the positive answers "yes". We will notice this phenomenon in the algorithms of both of the above-mentioned methods: if the formula F follows, indeed, from the formulas $A_1, \dots, A_n$, then algorithms will terminate and answer "yes", but otherwise the processing, instead of answering "no", frequently, will not terminate.

Note. We can try to improve the situation by exploring in parallel the queries G and $\neg G$. If our algorithm will answer "yes" for $\neg G$, that will mean the

answer “no” for G (if our the axioms stored in our knowledge base are consistent). But, if the axioms are *incomplete*, i.e., they do not allow to decide between G and $\neg G$, then the process will not terminate, nevertheless.

Both methods will start by applying one of the well known approaches to proving theorems in mathematics – and, it appears, it is especially convenient for computers as well – the so-called **refutation proofs** (*reductio ad absurdum*) – proofs by deriving a contradiction: assume $\neg G$, i.e., consider the set of formulas $A_1, \ldots, A_n, \neg G$ and try deriving a contradiction within it (Resolution Method). Or, equivalently (Theorem 4.3.5.(c)), try showing that the set $A_1, \ldots, A_n, \neg G$ is unsatisfiable simultaneously (Tableaux Method).

Re-read the text around Theorem 4.3.5 in Section 4.3. This will make reading of Section 6 more convenient.

Tableaux method was proposed by Evert Willem Beth and Jaakko Hintikka in 1955 and Kurt Schütte in 1956.

E. W. Beth (1955). Semantic entailment and formal derivability. *Mededelingen Koninklijke Nederlandse Akademie van Wetenschappen*, Nieuwe Reeks, 18, 13, 309-342.

K. J. J. Hintikka (1955). Form and content in quantification theory. *Acta Philosophica Fennica*, 8, 7–55.

K. Schütte (1956). Ein System des verknüpfenden Schließens. *Archiv für mathematische Logik und Grundlagenforschung*, Bd. 2 Heft 2–4, 34–67.

More about Beth – see EVERT W. BETH by Giorgio T. Bagni, Dipartimento di Matematica - Università di Torino.

The exposition below follows the elegant formulation of the method proposed by Raimond Smullian:

R. Smullyan (1968). First-Order Logic. Springer, New York.

See also Method of analytic tableaux in Wikipedia.

6.1. Tableaux Method for Propositional Formulas

Given the propositional formulas (i.e., formulas built of atoms by using logical connectives only) $A_1, \ldots, A_n, G$, we wish to establish, does G follow from $A_1, \ldots, A_n$, or not.

Of course, the case n=0 is included here: we may wish to establish, is G provable in the classical propositional logic, or not.

Phase 1. Instead of trying to build a proof

$$[L_1\text{-}L_{11}, MP]: \quad A_1, \ldots, A_n \vdash G,$$

let us **assume ¬G**, and consider the set of formulas $A_1, \ldots, A_n, \neg G$, trying to determine, is this set simultaneously satisfiable, or not.

If the set $A_1, \ldots, A_n, \neg G$ is *simultaneously satisfiable* (i.e., if there is an assignment of truth values to atoms such that all the $n+1$ formulas become true), then G *does not follow* from $A_1, \ldots, A_n$ (adding of ¬G does not cause contradictions).

If $A_1, \ldots, A_n, \neg G$ is *unsatisfiable simultaneously*, then G *follows* from $A_1, \ldots, A_n$ (adding of ¬G to $A_1, \ldots, A_n$ leads to contradictions).

Phase 2. Let us apply the method described in Section 5.1 to transform all the formulas $A_1, \ldots, A_n, \neg G$ into their **negation normal forms (NNFs)**, i.e., let us obtain a set of formulas equivalent to the respective initial formulas, that (the set) consists of atoms, conjunctions, disjunctions and negations (the latter are located at the atoms only). An example:

$$(\neg A \vee B) \wedge (\neg B \vee (\neg C \wedge D)) \wedge (A \vee B \vee C) \quad .$$

Because of the equivalence, the set of NNFs is true under the same truth-value assignments, as the initial set, In particular, the set of NNFs is simultaneously satisfiable, if and only so is initial set.

Phase 3. In order to reduce the satisfiability of our set of formulas to the satisfiability of their sub-formulas, let us **build a specific tree of formulas**. Indeed:

$A \wedge B$ is satisfiable, if A and B both are *simultaneously* satisfiable, so, we could follow the pattern: $\begin{matrix} A \wedge B \\ A \\ B \end{matrix}$.

$A \vee B$ is satisfiable, if A is satisfiable, or B is satisfiable, so, we could follow the branching pattern: $\begin{matrix} A \wedge B \\ A \quad B \end{matrix}$.

Before proposing a general algorithm, let us first consider an example, starting with the formula

$(A \vee \neg B) \wedge (B \vee \neg C)$.

This formula will be satisfied, if both of conjunction members will be satisfied:

$$\begin{matrix} (\boldsymbol{A \vee \neg B}) \wedge (\boldsymbol{B \vee \neg C}) \\ A \vee \neg B \\ B \vee \neg C \end{matrix}$$

Further, $A \vee \neg B$ will be satisfied, if one of the disjunction members will be satisfied:

$$(\boldsymbol{A \vee \neg B}) \wedge (\boldsymbol{B \vee \neg C})$$
$$\boldsymbol{A \vee \neg B}$$
$$B \vee \neg C$$
$$A \quad \neg B$$

$B \vee \neg C$ will be satisfied, if one of the disjunction members will be satisfied. But, since we must ensure that $A \vee \neg B$ and $B \vee \neg C$ are satisfied *simultaneously*, the branching to $B ; \neg C$ must be performed twice – after A, and after $\neg B$:

$$(\boldsymbol{A \vee \neg B}) \wedge (\boldsymbol{B \vee \neg C})$$
$$\boldsymbol{A \vee \neg B}$$
$$\boldsymbol{B \vee \neg C}$$
$$A \quad \neg B$$
$$B \quad \neg C \quad B \quad \neg C$$

This represents the entire tree that will be built by the tableaux method for the formula $(A \vee \neg B) \wedge (B \vee \neg C)$. It contains four branches:

1) In the first branch, assigning B=1 makes $B \vee \neg C$ true, an assigning A=1 makes $A \vee \neg B$ true, hence, all that makes $(A \vee \neg B) \wedge (B \vee \neg C)$ true.

2) In the second branch, assigning C=0 and A=1 does the same.

3) The third branch contains a *contradiction*: both B and $\neg B$.

4) In the fourth branch, assigning C=0 and B=0 make $(A \vee \neg B) \wedge (B \vee \neg C)$ true, like as in the first and second branches.

Thus, if, in a tree, all formulas are reduced to atoms, then, it seems, each of the branches that does not contain contradictions yields an assignment of truth values to atoms making the root formula true, i.e., *such a branch proves that the root formula is satisfiable.*

But what about *unsatisfiable* formulas? If our above guess is valid and every branch without contradictions makes the root formula true, then the tree built for an unsatisfiable formula must contain contradictions in every branch!

Let us consider an example of such a formula – the negation of the axiom L_8: $\neg[(B \to D) \to ((C \to D) \to (B \vee C \to D))]$, or, in the negation normal form: $(\neg B \vee D) \wedge (\neg C \vee D) \wedge (B \vee C) \wedge \neg D$. The corresponding tree looks as follows:

$\neg \boldsymbol{B} \vee \boldsymbol{D}$
$\neg \boldsymbol{C} \vee \boldsymbol{D}$
$\boldsymbol{B} \vee \boldsymbol{C}$
$\neg D$
$\neg B \quad D$
$\neg C \quad D \quad \neg C \quad D$
$B \quad C \; B \quad C \; B \quad C \; B \quad C$

Each of the eight branches contains a contradiction (verify). In fact, to conclude this, building of an incomplete smaller tree would be enough:

$\neg \boldsymbol{B} \vee \boldsymbol{D}$
$\neg \boldsymbol{C} \vee \boldsymbol{D}$
$\boldsymbol{B} \vee \boldsymbol{C}$
$\neg D$
$\neg B \quad D$
$\neg C \quad D \quad .$
$B \quad C \quad .$

So, our guess seems to be confirmed: the tree built for any unsatisfiable formula will, indeed, contain contradictions in every branch.

Now we can formulate the tree-building algorithm in full generality:

1) The **tree starts** with a single chain containing the negation normal forms of the formulas $A_1, \ldots, A_n, \neg G$. This corresponds to the processing rule for conjunctions (see below), because a set of formulas is meant as equivalent to the conjunction of all its formulas.

2) Now let us describe the **processing step**. Each step is processing of a single formula located in the tree. After this, the formula is marked as "processed" (in the above examples the processed formulas are marked bold). For the next step, one of the unprocessed formulas is selected.

2a) If the outer connective of the selected formula F is **conjunction**: $F_1 \wedge \ldots \wedge F_k$, then the sequential pattern $F_1, \ldots, F_k$ is appended to *each branch of the tree traversing the node* of the formula F.

2b) If the outer connective of the selected formula F is **disjunction**: $F_1 \vee \ldots \vee F_k$, then the branching pattern $\begin{array}{c} F_1 \vee \ldots \vee F_k \\ \hline F_1 \quad F_2 \quad \ldots \quad F_k \end{array}$ is appended to *each branch of the tree traversing the node* of the formula F.

3) The **tree is terminated**, if no formulas to be processed are left. At this moment, its nodes contain either processed formulas, or atoms (with or without negations). See the above examples.

Phase 4. Let us explore the branches of the tree and draw conclusions. If a

branch contains, for some atom A, both A and $\neg A$, such a branch is qualified as **closed**. Otherwise, i.e., if a branch contains, for every atom A, either A, or $\neg A$, or none of these, then it is qualified as **open**.

We will prove below Theorem 6.1.2 allowing us to conclude the following:

a) If the tree *contains an open branch*, then the set of formulas $A_1, \ldots, A_n, \neg G$ is **satisfiable simultaneously**, and hence, the formula G **does not follow from the formulas** $A_1, \ldots, A_n$. For the case n=0 this means that G is not provable in the classical logic.

b) If *all branches* of the tree *are closed*, then the set of formulas $A_1, \ldots, A_n, \neg G$ is **unsatisfiable simultaneously**, and hence, the formula G **follows from the formulas** $A_1, \ldots, A_n$. For the case n=0 this means that G is provable in the classical logic.

Note. The tableaux method, as described above, can be used also for **checking the consistency** of a set of formulas (the set of axioms of our knowledge base, for example). Indeed, the set is consistent, if and only of it is simultaneously satisfiable. Thus, we must simply start the process with the set $A_1, \ldots, A_n$. In Phase 4, the case (a) will correspond to the consistency of the set, the case (b) – to inconsistency.

Exercise 6.1.1. Use the tableaux method to determine:

a) Is the following formula provable in the classical propositional logic: $(B \to (C \to D)) \to ((B \to C) \to (B \to D))$?

b) Does the formula $B \to C$ follow from $B \vee D \to C \vee D$?

Now we will prove a theorem that will justify our conclusions in the Phase 4.

Lemma 6.1.1. (**Hintikka's Lemma 1**). Consider a finite set S of propositional formulas, all in the negation normal form. Assume, it possess the following properties:

– with every its member-formula $F_1 \wedge \ldots \wedge F_k$, it contains also all the formulas $F_1, \ldots, F_k$,

– with every its member-formula $F_1 \vee \ldots \vee F_k$, it contains also at least one of the formulas $F_1, \ldots, F_k$,

– for every atom A, it contains either A, or $\neg A$, or none of these (but not both).

Then the set S is simultaneously satisfiable, i.e., there is an assignment of truth-values to atoms making true all the formulas of S.

Proof. Let us assign the truth-values to all atoms appearing in S (with or without negation) in the following way: if A appears in it, then assign A=1, if

$\neg A$ appears, then assign A=0, if none of both appear, the truth value of A can be set arbitrary. Under this assignment (let us denote it by Γ), all the "atomic" members of S (i.e., atoms with or without negations) are true.

Let us verify, that all the other member-formulas of S are true under Γ as well. Indeed, if some members of S are false, let us consider among these false members the "smallest" one containing the minimum number of conjunctions and disjunctions. If this minimal false formula is $F_1 \wedge ... \wedge F_k$, then one of the formulas $F_1, ..., F_k$ must be false as well. This is impossible, because S contains all these "smaller" formulas. If the minimal false formula is $F_1 \vee ... \vee F_k$, then all the formulas $F_1, ..., F_k$ must be false as well. This is impossible, because S contains at least one of these "smaller" formulas.

Contradiction. Thus, all the member-formulas of S are true under Γ.

Q.E.D.

Theorem 6.1.2 (convergence, correctness and completeness of the tableaux method). For any set of propositional formulas, the tableaux method terminates being generated a finite tree of formulas such that:

a) If the tree contains an open branch, then the set is simultaneously satisfiable, and hence, consistent.

b) If all branches of the tree are closed, then the set is unsatifiable simultaneously, and, hence, inconsistent.

Proof. 1) Convergence. The method terminates for any set of propositional formulas. Indeed, each of the (finite number of) conjunctions an disjunctions contained in the negation normal forms of the formulas, is processed only once.

2) Assume, the generated tree contains an open branch O, and consider the set S of all formulas located at the nodes of O. Let us show that S satisfies all the conditions of Hintikka's Lemma 1. Then, according to this lemma, the set S is simultaneously satisfiable. And then, so is also a subset of S – the initial set of formulas for which the tree was generated.

Indeed, if S contains $F_1 \wedge ... \wedge F_k$, then, when processing this formula, all the formulas $F_1, ..., F_k$ will appear in every branch traversing $F_1 \wedge ... \wedge F_k$, hence, in the branch O as well. Thus, all of $F_1, ..., F_k$ belong to S.

If S contains $F_1 \vee ... \vee F_k$, then, when processing this formula, the branching pattern $\begin{matrix} F_1 \vee ... \vee F_k \\ F_1 \quad F_2 \quad ... \quad F_k \end{matrix}$ will appear in every branch traversing $F_1 \vee ... \vee F_k$, hence, one of the formulas $F_1, ..., F_k$ will appear on the branch O. This formula belongs to S.

Now, consider an atom A. Since O is an open branch, A and $\neg A$ cannot both appear on it, hence, they cannot both belong to S.

Q.E.D.

3) Now, on the other hand, assume that the initial set of formulas is simultaneously satisfiable, i.e., all these formulas are true for a single common assignment Γ of truth-values to atoms. Let us show that then, the tree generated by the tableaux method, contains an open branch.

Let us scan the generated tree in breadth-first mode, erasing all sub-trees, starting at the formulas that are *false* under the assignment Γ. At the root of the tree – all the initial formulas will remain, of course.

Now, consider any other moment: the scan process has reached a node containing a true formula H. All the other nodes in the branch until H, contain true formulas as well (see below). What could follow after H in the tree? Two situations are possible:

a) There follows a sequential pattern $F_1, \ldots, F_k$ generated when processing some true formula $F_1 \wedge \ldots \wedge F_k$ (in the branch of H, or H itself). Then all the formulas $F_1, \ldots, F_k$ are true es well. Continue scanning, i.e., process the next formula at the level of H (if any), or the left formula of the next level (if any).

b) There follows a branching pattern $\begin{array}{c} F_1 \vee \ldots \vee F_k \\ \hline F_1 \quad F_2 \quad \ldots \quad F_k \end{array}$ generated when processing some true formula $F_1 \vee \ldots \vee F_k$ (in the branch of H, or H itself). Then at least one of the formulas F_i is true as well, so, let us erase all the false formulas F_i together with their sub-trees. Continue scanning, i.e., process the next formula at the level of H (if any), or the left formula of the next level (if any).

After scanning-erasing is finished, the reduced tree cannot contain incomplete branches, i.e., branches that are extended in the original tree. Indeed, if, in the original tree, some branch is extended after a true formula H, then so is the branch in the reduced tree.

And all branches of the *reduced* tree are open – this tree consists of true formulas only, hence, contradictions in its branches are impossible. Hence, the *initial* tree contains at least one open branch.

Q.E.D.

Possible improvements

1. When generating a tree, after inserting a formula F (or $\neg F$) into it, we might notice that the branch already contains $\neg F$ (or, F). Usually, this

happens when F is an atom. In any such case, we can stop extending the branch by marking it as already closed. This could save both generation time and space.

2. If there is a choice, process conjunctions $F_1 \wedge ... \wedge F_k$ first, processing disjunctions $F_1 \vee ... \vee F_k$ as late as possible. This could make the resulting tree smaller.

Computational complexity

The time and space used by the tableaux method depends mainly on the size of the tree that is generated for a particular set of formulas $A_1, ..., A_n, \neg G$. Estimating roughly, the size of the tree is most affected by the number d of disjunctions in NNFs of the given formulas. In **the worst case**, the number of nodes in the tree does not exceed $p(l)2^d$, where l is the total length of the formulas $A_1, ..., A_n, \neg G$, and p – a polynomial.

Exercise 6.1.2 (optional, for smart students). Verify this.

This estimate corresponds well to the general complexity problem, mentioned at the end of Section 4.2: the problem of determining, is a propositional formula provable in the classical propositional logic, or not, belongs to the complexity class co-NP-complete. Of course, tableaux method cannot overcome this problem.

But in many **practical situations**, experience shows that tableaux method solves its task in an acceptable time.

6.2. Tableaux Method for Pure Predicate Formulas

Attention! Tableaux method applies only to the so-called **pure predicate languages.**

Such languages *do not contain functions constants*, thus, their only kinds of **terms** are object variables and object constants (if any).

Given a set of formulas $A_1, ..., A_n, G$ in some pure predicate language, we wish to establish, does G follow from $A_1, ..., A_n$, or not.

Of course, the case n=0 is included here: trying to establish, is G provable in the classical logic, or not.

Of course, solving of our new task will be more complicated when compared with the case of propositional formulas considered in the previous section.

Phase 1. Instead of trying to build a proof

$$[L_1\text{-}L_{15}, MP, Gen]: \quad A_1, \dots, A_n \vdash G,$$

let us **assume ¬G,** and consider the set of formulas $A_1, \dots, A_n, \neg G$, trying to determine, is this set simultaneously satisfiable, or not.

If the set $A_1, \dots, A_n, \neg G$ is *simultaneously satisfiable* (i.e., if there is an interpretation under which all the $n+1$ formulas are true), then G *does not follow* from $A_1, \dots, A_n$ (adding of ¬G does not cause contradictions).

If $A_1, \dots, A_n, \neg G$ is *unsatisfiable simultaneously*, then G *follows* from $A_1, \dots, A_n$ (adding of ¬G to $A_1, \dots, A_n$ leads to contradictions).

For predicate formulas, the following additional step is necessary in Phase 1: if some of the formulas $A_1, \dots, A_n, \neg G$ contain free variables, for example, $x_1, \dots, x_m$, then replace the entire set by a single formula

$$\exists x_1 \dots \exists x_m (A_1 \wedge \dots \wedge A_n \wedge \neg G) \quad . \qquad (*)$$

Exercise 6.1.3. Verify that this formula is satisfiable if and only if the set $A_1, \dots, A_n, \neg G$ is simultaneously satisfiable.

Thus, we can assume that all the formulas to be processed are **closed formulas**.

Phase 2. Let us apply the method described in Section 5.1 to transform all the formulas $A_1, \dots, A_n, \neg G$ (or the formula (*)) into their **negation normal forms (NNFs)**, i.e., let us obtain a set of formulas, equivalent to the respective initial formulas, that consist of atomic formulas, quantifiers, conjunctions, disjunctions and negations (the latter are located at the atomic formulas only). An example:

$$\forall x (\neg p(x) \vee \exists y (q(y) \wedge \forall z (\neg r(x, z) \vee s(z)))) \quad .$$

Because of the equivalence, the set of NNFs is true under the same interpretations as the initial set. In particular, the set of NNFs is simultaneously satisfiable, if and only so is initial set.

Phase 3. In order to reduce the satisfiability of our set of formulas to the satisfiability of their sub-formulas, let us **build a specific tree of formulas**.

Let us formulate the tree-building algorithm at once in full generality:

1) The **tree starts** with a single chain containing the negation normal forms of the formulas $A_1, \dots, A_n, \neg G$ (or, with the negation normal form of the formula (*)).

2) Now let us describe the **processing step**. Each step is processing a single formula located in the tree. After this, the formula is marked as “processed”.

For the next step, one of the unprocessed formulas is selected.

The first two variants of the step coincide with the propositional case.

2a) If the outer connective of the selected formula F is **conjunction**: $F_1 \wedge ... \wedge F_k$, then the sequential pattern $F_1, ..., F_k$ is appended to *every branch of the tree traversing the node* of the formula F. Motivation: to make $F_1 \wedge ... \wedge F_k$ true, we must first ensure that $F_1, ..., F_k$ are true all simultaneously.

2b) If the outer connective of the selected formula F is **disjunction**: $F_1 \vee ... \vee F_k$, then the branching pattern $\begin{array}{cccc} F_1 \vee ... \vee F_k & & & \\ F_1 & F_2 & ... & F_k \end{array}$ is appended to *every branch of the tree traversing the node* of the formula F. Motivation: to make $F_1 \vee ... \vee F_k$ true, we must ensure that at least one formula of $F_1, ..., F_k$ is true.

The remaining two variants are new.

2c) If the selected formula is an **existential** one – $\exists x F(x)$, then a new object constant c is introduced, and the formula $F(c)$ is appended to *every branch of the tree traversing the node* of the formula $\exists x F(x)$. "New" means here that c does not appear in the branch, ending in $\exists x F(x)$. Motivation: if, in some interpretation $F(c)$ will be made true, then so will be $\exists x F(x)$.

2d) If the selected formula is a **universal** one – $\forall x F(x)$, then the following sequential pattern is appended to *every branch of the tree traversing the node* of the formula $\forall x F(x)$:

$$F(c), F(d), ..., \forall x F(x) \quad .$$

Here: c, d, ... are *all* the object constants that appear in the branch, and for which, the instances $F(c), F(d), ...$ have not yet been placed in the branch. (If no object constants appear in the branch ending in $\forall x F(x)$, then a new constant c is introduced. See Example 3 below how this works.) As we see, here, the formula $\forall x F(x)$ is repeated at the end of the pattern. Motivation: to make $\forall x F(x)$ true, we must ensure that F(c) is true at least for all the constants appearing in the branch. And, since new constants may appear in the branch over and again, $\forall x F(x)$ must be repeated over and again as well!

3) A **branch is marked as terminated** (i.e., closed) immediately, when a contradiction appears in it. The **tree is terminated**, if all of its branches are marked as terminated, or, if no formulas to be processed are left. Unfortunately, because of the steps (2d) repeatedly processing universal quantifiers, in some cases, the termination may not occur. Then, we can only imagine "termination in infinity" and infinite branches in the tree.

However, we will prove below that, if the initial set of formulas is *unsatisfiable simultaneously,* then the tree will terminate *after a finite number of steps*, with contradictions in all of its branches (i.e., with all branches closed).

Phase 4. Let us imagine, we are exploring the branches of the tree and try drawing conclusions. Sometimes, we can only imagine this – because, as noted above, some of the branches may grow infinitely.

We will prove below Theorem 6.2.2 allowing us to conclude the following:

a) Assume, the set $A_1, \dots, A_n, \neg G$ is unsatisfiable simultaneously, i.e., the formula G **follows from the formulas** $A_1, \dots, A_n$ (for the case n=0 this means that G is provable in the classical logic). Then and only then, the tree terminates *after a finite number of steps*, with contradictions in all of its branches (i.e., with all branches closed).

Hence, if the formula G follows from the formula**s** $A_1, \dots, A_n$, the tableaux method allows to establish this **after a finite number of steps**.

b) If the tree contains an open (finite or **infinite**) branch, then the set of formulas $A_1, \dots, A_n, \neg G$ is simultaneously **satisfiable**, and hence, the formula G **does not follow from the formulas** $A_1, \dots, A_n$. For the case n=0 this means that G is not provable in the classical logic.

Unfortunately, not always, the situation (b) can be recognized after a finite number of steps.

Example 1. Let n=0 and G be $\exists x\, p \vee \exists x\, q \rightarrow \exists x (p \vee q)$.

Negation normal form of $\neg G$: $(\exists x\, p \vee \exists x\, q) \wedge \forall x (\neg p \wedge \neg q)$.

Let the tableaux method to start processing:

$$\begin{array}{cc} \exists x\, p \vee \exists x\, q & \\ \forall x (\neg p \wedge \neg q) & \\ \exists x\, p & \exists x\, q \\ p(c_1) & q(c_1) \\ \neg p(c_1) \wedge \neg q(c_1) & \neg p(c_1) \wedge \neg q(c_1) \\ \neg p(c_1) & \neg q(c_1) \end{array}$$

No need to continue: both branches are closed already. Thus, $\neg G$ is unsatisfiable, and G – **provable** in the classical logic (compare with Section 3.3).

Example 2. Let n=0 and G be $\exists x\, p \wedge \exists x\, q \rightarrow \exists x (p \wedge q)$.

Negation normal form of $\neg G$: $(\exists x\, p \wedge \exists x\, q) \wedge \forall x (\neg p \vee \neg q)$.

Let us start processing:

$$\exists x\, p \wedge \exists x\, q$$
$$\forall x(\neg p \vee \neg q)$$
$$\exists x\, p$$
$$\exists x\, q$$
$$p(c_1)$$
$$q(c_2)$$
$$\neg p(c_1) \vee \neg q(c_1)$$
$$\neg p(c_2) \vee \neg q(c_2)$$
$$\boldsymbol{\forall x(\neg p \vee \neg q)}$$

$\neg p(c_1)$ $\neg q(c_1)$

closed $\neg p(c_2)$ $\neg q(c_2)$

closed

Formally, in the branch after $\neg p(c_2)$, we must process the marked formula $\forall x(\neg p \vee \neg q)$ again, but in fact, this is not necessary: no new constants have appeared in the branch. Thus, we see that our tree contains an open branch. Hence, the formula $\neg G$ is satisfiable, and **G cannot be proved** in the classical logic – "as it should be" (explain, why).

From the open branch, we can derive an interpretation making $\neg G$ true. Indeed, take: $D=\{c_1, c_2\}; p(c_1)=1, q(c_2)=1, q(c_1)=0, p(c_2)=0$ (verify).

Here we have the happy situation being able to detect an open branch after a finite number of steps. In general, this is impossible – because of the Unsolvability Theorem. There are formulas that can be satisfied only in infinite domains. The tableaux-trees of such formulas will contain infinite open branches only.

Exercise 6.2.1 (optional, for smart students)**.** Construct a formula that can be satisfied only in an nfinite domain. Try processing with the tableaux method.

Example 3. Let n=0 and G be $\forall x\, p(x) \rightarrow \exists x\, p(x)$ **.**

Negation normal form of $\neg G$: $\forall x\, p(x) \wedge \forall x \neg p(x)$.

Let us start processing:

$$\forall x\, p(x)$$
$$\forall x \neg p(x)$$
$$p(c_0)$$
$$\neg p(c_0)$$

No need to continue: the only branch is closed already. Thus, $\neg G$ is unsatisfiable, and G – **provable** in the classical logic (compare with Section 3.1). As noted above in the description of step (2d): to enable processing of the universal quantifier, we were forced here to introduce a new object constant.

Exercise 6.2.2. Use the tableaux method to determine which of the following formulas are provable in the classical logic (G does not contain x as a free variable):

a) $\forall x(F(x)\rightarrow G)\rightarrow(\exists x\,F(x)\rightarrow G)$;

b) $\exists x(p\wedge q)\rightarrow\exists x\,p\wedge\exists x\,q$?

Now let us prove the theorem that will justify our conclusions in the Phase 4.

Lemma 6.2.1. (**Hintikka's Lemma 2**). Consider a (finite or infinite) set S of closed formulas in some predicate language, all in the negation normal form. Assume, it possess the following properties:

– with every its member-formula $F_1\wedge\ldots\wedge F_k$, it contains also all the formulas $F_1,\ldots,F_k$,

– with every its member-formula $\exists x\,F(x)$ and $\forall x\,F(x)$, it contains also at least one formula $F(c)$, where c is an object constant,

– with every its member-formula $\forall x\,F(x)$, it contains also at least one formula F(c), and all the formulas $F(c)$ for all object constants appearing in S,

– for every atomic formula $p(c_1,\ldots,c_k)$, where p is a predicate constant, and c_i are object constants, it contains either $p(c_1,\ldots,c_k)$, or $\neg p(c_1,\ldots,c_k)$, or none of these (but not both).

Then, the set S is simultaneously satisfiable, i.e., there is an interpretation of the language making true all the formulas of S.

Proof. Let us take as the domain of the desired interpretation the set D of all object constants appearing in S. And let us assign the truth-values to predicate constants in the following way:

if an atomic formula $p(c_1,\ldots,c_k)$ appears in S, then set $p(c_1,\ldots,c_k)=1$,

if, instead, $\neg p(c_1,\ldots,c_k)$ appears, then set $p(c_1,\ldots,c_k)=0$,

if none of both appear, the truth-value of $p(c_1,\ldots,c_k)$ can be set arbitrary.

Note. If the set S is infinite, we may never get to know, when to apply the third of the above rules. In this situation, we can guarantee only that the interpretation of the predicate constant p is an **algorithmically** (recursively) **enumerable predicate**, but cannot guarantee that this predicate is algorithmically solvable (recursive, computable).

Under the above interpretation (let us denote it by J), all the "atomic" members of S (i.e., atomic formulas with or without negations) are true.

Let us verify, that all the other member-formulas of S are true under J as well. Indeed, if some members of S are false under J, let us consider among these false members the "smallest" one containing the minimum number of

conjunctions, disjunctions and quantifiers.

If this minimal false formula is $F_1 \wedge ... \wedge F_k$, then one of the formulas $F_1, ..., F_k$ must be false as well. This is impossible, because S contains all these "smaller" formulas.

If the minimal false formula is $F_1 \vee ... \vee F_k$, then all the formulas $F_1, ..., F_k$ must be false as well. This is impossible, because S contains at least one of these "smaller" formulas.

The minimal false formula cannot be $\exists x F(x)$, because S contains at least one formula $F(c)$, which is true.

The minimal false formula cannot be $\forall x F(x)$ as well, because S contains at least one formula $F(c)$, and all formulas $F(c)$ in S are true.

Contradiction. Thus, all the member-formulas of S are true under J.

Q.E.D.

Theorem 6.2.2 (correctness, completeness and semi-convergence of the tableaux method). For any set of formulas in some predicate language, the tableaux method generates a finite or infinite tree of formulas such that:

a) If the set is unsatisfiable simultaneously, then the method **terminates** after a finite number of steps, the generated tree is **finite**, and all of its branches are **closed**.

b) If the set is simultaneously satisfiable, then the generated tree contains at least one, finite or infinite, **open** branch.

Proof. 1) First, assume that the generated tree contains an open (finite or infinite) branch O, and consider the set S of all formulas located at the nodes of O. Let us show that S satisfies all the conditions of Hintikka's Lemma 2. Then, according to this lemma, the set S is simultaneously satisfiable. And then, so is also a subset of S – the initial set of formulas for which the tree was generated.

Indeed, if S contains $F_1 \wedge ... \wedge F_k$, then, when processing this formula, all the formulas $F_1, ..., F_k$ will appear in every branch traversing $F_1 \wedge ... \wedge F_k$, hence, in the branch O as well. Thus, all of $F_1, ..., F_k$ belong to S.

If S contains $F_1 \vee ... \vee F_k$, then, when processing this formula, the branching pattern $\begin{matrix} F_1 \vee ... \vee F_k \\ F_1 \quad F_2 \quad ... \quad F_k \end{matrix}$ will appear in every branch traversing $F_1 \vee ... \vee F_k$, hence, one of the formulas $F_1, ..., F_k$ will appear on the branch O. Then, this formula belongs to S.

If S contains $\exists x F(x)$, then, when processing this formula, some formula $F(c)$ will appear in every branch traversing $\exists x F(x)$, hence, in the branch O as well. This formula belongs to S.

If S contains $\forall x F(x)$, then, processing this formula repeatedly, the formulas $F(c)$, for all the object constants c appearing in O will appear in every branch traversing $\forall x F(x)$, hence, in the branch O as well. Thus, all these formulas belong to S.

Now, consider an atomic formula $p(c_1, \ldots, c_k)$. Since O is an open branch, $p(c_1, \ldots, c_k)$ and $\neg p(c_1, \ldots, c_k)$ cannot both appear on it, hence, they cannot both belong to S. Q.E.D.

2) Now, on the other hand, assume that the initial set of formulas is simultaneously satisfiable, i.e., all these formulas are true under some interpretation J. Let us show that then, the tree generated by the tableaux method contains an open branch.

Since each level of the generated tree contains a finite number of nodes, let us scan the entire tree in the breadth-first mode, level by level, erasing all sub-trees, that are starting at the formulas that are *false* under J. At the root of the tree – all the initial formulas will remain, of course.

The interpretations of all the object constants appearing in the initial formulas, are already defined in J.

Now, consider any other moment: the scan process has reached a node containing a true formula H. All the other nodes in the branch above H, contain true formulas as well. What could follow immediately after H in the tree? Four situations are possible:

a) There follows a sequential pattern $F_1, \ldots, F_k$ generated when processing some true formula $F_1 \wedge \ldots \wedge F_k$ (in the branch above H, or H itself). Then all the formulas $F_1, \ldots, F_k$ are true es well. Thus, H is immediately followed by true formulas only. Continue scanning, i.e., process the next formula at the level of H (if any), or the left formula of the next level (if any).

b) There follows a branching pattern $\begin{array}{c} F_1 \vee \ldots \vee F_k \\ F_1 \quad F_2 \quad \ldots \quad F_k \end{array}$ generated when processing some true formula $F_1 \vee \ldots \vee F_k$ (in the branch above H, or H itself). Then at least one of the formulas F_i is true as well, so, let us erase all the false formulas F_i together with their sub-trees. Thus, after erasing, H is immediately followed by true formulas only. Continue scanning, i.e., process the next formula at the level of H (if any), or the left formula of the next level (if any).

c) There follows the formula $F(c)$ generated when processing some true

formula $\exists x F(x)$ (in the branch above H, or H itself). Since the object constant c was introduced as *new for the entire branch*, we can freely define the interpretation of c as one of the objects of J making $\exists x F(x)$ true. Then the formula $F(c)$ will be true as well. Thus, H is immediately followed by a true formula. Continue scanning, i.e., process the next formula at the level of H (if any), or the left formula of the next level (if any).

d) There follows a sequence of formulas

$$F(c), F(d), \ldots, \forall x F(x)$$

generated, when processing some true formula $\forall x F(x)$ (in the branch above H, or H itself). Here: c, d, ... are *all* the object constants that appear in the branch of H, and for which, the instances $F(c), F(d), \ldots$ have not yet be placed in the branch. The interpretations of c, d,… are already defined earlier. If no object constants appear in the branch ending in $\forall x F(x)$, then a new constant c was introduced, the interpretation of it now can be defined freely as any object of J (all of them make $F(x)$ true). Thus, H is followed by true formulas only.

After scanning-erasing is finished (maybe, “in the limit” only), the reduced tree contains only formulas that are true under the interpretation J, hence, contradictions in its branches are impossible. Thus, all branches of such a tree (finite or infinite ones) are open. And such a tree cannot contain incomplete branches, i.e., branches that are extended in the original tree. Indeed, if, in the original tree, some branch is extended after a true formula H, then so is extended the branch of the reduced tree. Hence, the *initial tree* contains at least one open branch. Q.E.D.

3) Thus, we have proved that the initial set of formulas is simultaneously satisfiable if and only if, the tree generated by the tableaux method, contains a (finite or infinite) open branch.

Hence, the initial set is unsatisfiable simultaneously if and only if, all branches of the generated tree are closed. A *closed branch is finite* – it is marked as closed immediately when a contradiction appears in its nodes. According to Kőnig’s Lemma, an infinite finitely branching tree contains an infinite branch.

See Dénes Kőnig in Wikipedia.

The generated tree is finitely branching, indeed (see above the only branching steps (2b)). Hence, if the initial set is unsatisfiable simultaneously, then the generated tree is finite – and, of course, this can be established after a finite number of steps.

If the initial set is simultaneously satisfiable, then the generated tree contains an open (finite or infinite) branch. In this case, the process may or may not terminate. Q.E.D.

Computational complexity

The latter conclusions correspond well to the general complexity problem, mentioned at the end of Section 4.3: by the Unsolvability Theorem, the problem of determining, is a predicate formula provable in the classical predicate logic, or not, is *not algorithmically solvable*. Tableaux method represents a universal algorithm that terminates and answers correctly "yes" in all positive cases (G follows from $A_1, \ldots, A_n$, i.e., the initial set $A_1, \ldots, A_n, \neg G$ is unsatisfiable simultaneously). In the negative cases (G does not follow from $A_1, \ldots, A_n$, i.e., the initial set $A_1, \ldots, A_n, \neg G$ is simultaneously satisfiable) the algorithm may terminate and answer correctly "no", but it may not terminate as well.

But experience shows that in many **practical situations**, the tableaux method solves its task, and – in an acceptable time.

Attention: non-constructive reasoning! If the tree generated by the tableaux method and contains a *finite* open branch, then, from this branch, a finite interpretation can be extracted that is making true all the formulas of the initial set. However, if the open branch is *infinite*, then the interpretation extracted from it, may have in infinite domain of objects, and – correspondingly – the interpretations of the predicate constants may be or may not be computable (see the above note about the algorithmic enumerability). To understand this in detail – do Exercise 4.3.5 in Section 4.3.

7. Resolution Method

Attention! The principal results of this Section are valid only for the **classical logic**!

To start, re-read the introductory part of Section 6.

Main steps

Let us consider only **refutation proofs** as a means to derive consequences and prove theorems. Thus, in order to prove that some formula G follows from a set of formulas F_1, ..., F_n, let us add ¬G to the set and try deriving a contradiction.

Let us try developing a practical method of deriving contradictions from an inconsistent set of assumptions. This (at first glance – trivial) decision is one of the most important steps in the whole story – it will allow for conversion of the formulas F_1, ..., F_n, ¬G into a form that does not contain existential quantifiers (Skolem normal form, see Section 5.4). And after this, having universal quantifiers only, we will simply drop them at all, and continue working with quantifier-free formulas only (clause forms, see Section 5.5).

Thus, when trying to prove that F_1, ..., F_n, ¬G is an inconsistent set of formulas, let us first "normalize" these formulas as far as possible.

The first step is reducing into the so-called **prenex normal form** – moving all the quantifiers to left. For example, the formula

$$((\exists x\, B(x) \rightarrow \exists x\, C(x)) \rightarrow \exists x\, D(x)) \rightarrow \exists x\, F(x)$$

is equivalent (in the classical logic) to the following formula in prenex normal form:

$$\forall x_1 \exists x_2 \forall x_3 \exists x_4 (((B(x_1) \rightarrow C(x_2)) \rightarrow D(x_3)) \rightarrow F(x_4)) \quad .$$

When moving quantifiers to left, some of them must be changed from ∃ to ∀, or from ∀ to ∃ (see Section 5.3).

The second step allows elimination of existential quantifiers. Indeed, $\forall x_1 \exists x_2$ means that $x_2 = f(x_1)$, and $\forall x_1 \forall x_3 \exists x_4$ means that $x_4 = g(x_1, x_3)$, where f and g are some functions (see Section 5.4). In this way we obtain the so-called **Skolem normal form**, containing universal quantifiers only:

$$\forall x_1 \forall x_3((B(x_1) \rightarrow C(f(x_1)) \rightarrow D(x_3)) \rightarrow F(g(x_{1,} x_3))) \quad .$$

Note that a formula and its Skolem normal form **are not equivalent** in the classical logic, they are only "loosely equivalent": a set of formulas allows deriving of contradictions if and only if the set of their Skolem normal forms allows deriving of contradictions as well (and conversely).

Now, since, our formula, in its prefix, contains universal quantifiers only, we may **drop these quantifiers**:

$$(B(x_1) \rightarrow C(f(x_1)) \rightarrow D(x_3)) \rightarrow F(g(x_{1,} x_3)) \quad .$$

Nothing is lost here. The possibility of deriving contradictions is retained – the quantifiers can be restored by applying the *Gen*-rule.

The third step – reduction of quantifier-free formulas to the so-called **conjunctive normal form** (CNF, a conjunction of disjunctions of atomic formulas – with or without negations, see Section 5.2). For example, the above formula can be reduced to the following form:

$$(\neg B(x_1) \vee C(f(x_1)) \vee F(g(x_{1,} x_3))) \wedge (\neg D(x_3) \vee F(g(x_{1,} x_3))) \quad .$$

By assuming that a set of formulas means a conjunction of these formulas, we can drop the conjunction(s) obtaining a set of the so-called **clauses**:

$$\neg B(x_1) \vee C(f(x_1)) \vee F(g(x_{1,} x_3)) \quad ;$$
$$\neg D(x_3) \vee F(g(x_{1,} x_3)) \quad .$$

Each clause is a disjunction of atomic formulas – with or without negations.

To separate clearly the meaning of each clause, let us note that $\forall x(B(x) \wedge C(x)) \leftrightarrow \forall x\, B(x) \wedge \forall y\, C(y)$ (see Section 3). Thus, we can rename some of the variables – and no two clauses will contain common variables:

$$\neg B(x_1) \vee C(f(x_1)) \vee F(g(x_{1,} x_3)) \quad ;$$
$$\neg D(x_5) \vee F(g(x_{4,} x_5)) \quad .$$

In this way, instead of our initial set of assumptions F_1, ..., F_n, $\neg G$, we obtain a set of separate clauses, which allows for deriving of contradictions if and only if the initial set F_1, ..., F_n, $\neg G$ allows for it.

The last step is working with the obtained set of clauses – “a large cloud of simple disjunctions”.

It appears that a set of clauses allows deriving of contradictions if and only if a contradiction can be derived from it by using **term substitution** and the so-called **resolution rule**:

$$\frac{F \vee \boldsymbol{C}, \neg \boldsymbol{C} \vee G}{F \vee G} \ .$$

Details below.

History

J. A. Robinson. Theorem-proving on the computer. "Jour. Assoc. Comput. Mach.", vol.10, N2, 1963, pp.163-174

J. A. Robinson. A machine-oriented logic based on the resolution principle, "Jour. Assoc. Comput. Mach.", vol.12, N1, January 1965, pp.23-41 (available online, Russian translation available: "Kib. sbornik (novaya seriya)", 7, 1970, pp.194-218)

John Alan Robinson: "Born in Yorkshire in 1930, Robinson came to the United States in 1952 with a classics degree from Cambridge University. He studied philosophy at the University of Oregon before moving to Princeton where he received his PhD in philosophy in 1956. Temporarily ``disillusioned with philosophy,`` he went to work as an operations research analyst for Du Pont, where he learnt programming and taught himself mathematics. Robinson moved to Rice University in 1961, spending his summers as a visiting researcher at the Argonne National Laboratory's Applied Mathematics Division. Its then Director, William F. Miller, pointed Robinson in the direction of theorem proving...

Miller showed Robinson a 1960 paper by Martin Davis and Hilary Putnam (coincidentally, the latter had been Robinson's PhD supervisor) proposing a predicate-calculus proof procedure that seemed potentially superior to Gilmore's, but which they had not yet turned into a practical computer program. Miller suggested that Robinson use his programming skills to implement Davis and Putnam's procedure on the Argonne IBM 704. Robinson quickly found that their procedure remained very inefficient. However, while implementing a different procedure also suggested in 1960 by Dag Prawitz, Robinson came to see how the two sets of ideas could be combined into a new, far more efficient, automated proof procedure for first-order predicate logic: "resolution"..." (According to **D. A. MacKenzie**, The Automation of Proof: A Historical and Sociological Exploration, "IEEE Annals of the History of Computing", vol.17, N3, 1995, pp. 7-29).

Almost at the same time when J. A. Robinson invented the resolution method, Sergei Yurievich Maslov invented his **inverse method**, which has a similar range of applications:

S. Yu. Maslov. An inverse method of establishing deducibilities in the classical predicate calculus, "Soviet Mathematics, Doklady", 1964, N5, pp.1420-1423.

See also: Maslov S. Y. (1939-1982), human rights activist in ENCYCLOPAEDIA OF SAINT PETERSBURG.

About the history of the problem see:

J. A. Robinson. Computational Logic: Memories of the Past and Challenges for the Future. *Computational Logic – CL 2000, First International Conference, London, UK, 24-28 July, 2000, Proceedings*, Springer, Lecture Notes in Computer Science, 2000, Vol. 1861, pp. 1-24 (online copy).

M. Davis. The Early History of Automated Deduction.In: *Handbook of Automated Reasoning*, ed. by A. Robinson and A. Voronkov, Elsevier Science, 2001, vol. I, pp. 3-15.

7.1. Resolution Method for Propositional Formulas

The Method

Propositional formulas are built of atoms by using logical connectives only. Accordingly, their clause forms consist of disjunctions of atoms with or without negations.

How to work with a “cloud” of such disjunctions efficiently?

Assume that, in a set of clauses, two clauses are contained such that an atom C appears as a positive member in the first clause, and as a negative member in the second one:

$$\neg A_1 \vee \neg A_2 \vee ... \vee \neg A_m \vee B_1 \vee B_2 \vee ... \vee B_n \vee \boldsymbol{C} \quad , \qquad (1)$$

$$\neg \boldsymbol{C} \vee \neg D_1 \vee \neg D_2 \vee ... \vee \neg D_p \vee E_1 \vee E_2 \vee ... \vee E_q \quad , \qquad (2)$$

or, simply,

$$F \vee \boldsymbol{C} \quad , \qquad (1a)$$

$$\neg \boldsymbol{C} \vee G \quad . \qquad (2a)$$

If C is false, then (1a) yields F, and, if C is true, then (2a) yields G. Thus, from (1a) and (2a) we have derived $F \vee G$, i.e., deriving of $F \vee G$ from $F \vee \boldsymbol{C}$ and $\neg \boldsymbol{C} \vee G$ is "logically correct", and it is called the **resolution rule** (J. A. Robinson proposed to use it in the above 1963 paper):

$$\frac{F \vee C \, , \neg C \vee G}{F \vee G} \quad .$$

Taking into account the rule (of the classical logic) $\neg A \vee B \leftrightarrow (A \rightarrow B)$, we can obtain an alternative form of the resolution rule:

$$\frac{\neg F \rightarrow C \, , C \rightarrow G}{\neg F \rightarrow G} \quad .$$

In the classical logic, this form is equivalent to the Law of Syllogism (transitivity of implication).

If F is empty, then this form derives G from C and C→G, i.e., **resolution rule includes Modus Ponens** as a special case.

If G is empty, then from $\neg F \vee C \, , \neg C$ (i.e., F→C, ¬C), the **resolution rule** derives ¬F, i.e., it **includes Modus Tollens** as a special case.

Exercise 7.1.1. a) Derive the resolution rule in the constructive logic, i.e., prove that [L_1-L_{10}, MP]: $F \vee C \, , \neg C \vee G \vdash F \vee G$.

b) (optional, for smart students) Verify that it cannot be proved in the minimal logic [L_1-L_9, MP]. (Hint: in the positive part – use Theorem 2.5.1(b) [L_1, L_2, L_8, L_{10}, MP]: $F \vee C, \neg C \vdash F$. In the negative part – verify that in the minimal logic, the resolution rule allows proving of L_{10}, see Section 2.5).

Thus, from the clauses (1) and (2), resolution rule allows deriving of the following clause:

$$\neg A_1 \vee \neg A_2 \vee ... \vee \neg A_m \vee \neg D_1 \vee \neg D_2 \vee ... \vee \neg D_p \vee B_1 \vee B_2 \vee ... \vee B_n \vee E_1 \vee E_2 \vee ... \vee E_q$$

At first glance, this approach leads to nothing, because this formula seems to be much longer than (1), and than (2). Still, this is not 100% true, because, additionally, we can reduce the repeating atoms, and, finally, the set of different atoms, used in a clause form, is fixed! If, in our set of clauses, there are N different atoms, then none of the clauses (initial, or generated by resolution rule) will contain more than N atoms (each with or without negation). And the total number of different clauses will never exceed 3^N (*missing, without negation, with negation*). Thus, repeated applications of the resolution rule will "rotate" within this restricted "search space".

The smart idea behind the resolution rule is as follows: it is a **universal tool for deriving of contradictions from inconsistent sets of clauses**! No other axioms and rules of inference are necessary! More precisely, it is universal, if used together with the following trivial rules of inference:

$$\frac{F \vee \boldsymbol{C} \vee \boldsymbol{D} \vee G}{F \vee \boldsymbol{D} \vee \boldsymbol{C} \vee G} \quad \text{(permutation)},$$

$$\frac{F \vee \boldsymbol{C} \vee \boldsymbol{C} \vee G}{F \vee \boldsymbol{C} \vee G} \quad \text{(reduction)}.$$

The permutation rule allows for arbitrary reordering of atoms in a clause (for example, moving C to the right, and moving ¬C to the left). The reduction rule allows for reduction of repeating identical atoms.

Exercise 7.1.2. Derive these inference rules in the minimal logic, namely, prove that:

a) [L_1-L_8, MP]: $F \vee \boldsymbol{C} \vee \boldsymbol{D} \vee G \vdash F \vee \boldsymbol{D} \vee \boldsymbol{C} \vee G$.

b) [L_1-L_8, MP]: $F \vee \boldsymbol{C} \vee \boldsymbol{C} \vee G \vdash F \vee \boldsymbol{C} \vee G$.

Theorem 7.1.1 (J. A. Robinson). In the classical propositional logic [L_1-L_{11}, MP], a finite set of propositional clauses is inconsistent if and only if the resolution rule (together with permutation and reduction rules) allows for deriving of a contradiction from it.

Note. In some other texts, this fact is called "the refutation-completeness of the resolution

rule" for the propositional logic.

Proof. 1. As you have proved in the Exercises 7.1.1 and 7.1.2, all the formulas, derived from a set of formulas $K_1, K_2, \dots, K_s$ by using the permutation, resolution and reduction rules are consequences of $K_1, K_2, \dots, K_s$. Hence, if these rules allow deriving a contradiction from this set of formulas, then it (the set) is inconsistent.

2. Now, let us assume that a set of propositional clauses $K_1, K_2, \dots, K_s$ is inconsistent, i.e., a contradiction $A \wedge \neg A$ can be derived from it:

$$[L_1\text{-}L_{11}, MP]: \quad K_1, K_2, \dots, K_s \vdash A \wedge \neg A \quad .$$

Then, under the classical truth tables, the conjunction $K_1 \wedge K_2 \wedge \dots \wedge K_s$ takes only false values (Completeness Theorem). Let us mark one of the atoms (the atom C) in it. Let us denote:

- by $C \vee F_i$ – the clauses containing C without negation,

- by $\neg C \vee G_j$ – the clauses containing C with negation,

- by H_k – the clauses that do not contain C.

All the formulas F_i, G_j, H_k are disjunctions of atoms (with or without negations) that do not contain the atom C.

Thus $K_1 \wedge K_2 \wedge \dots \wedge K_s$ is equivalent to

$$conj(C \vee F_i) \wedge conj(\neg C \vee G_j) \wedge conj(H_k) \quad . \qquad (4)$$

Let us apply (the strange) one of the logical distribution rules:

$$[L_1\text{-}L_8, MP] \vdash (A \wedge B) \vee C \leftrightarrow (A \vee C) \wedge (B \vee C) \quad .$$

Hence, $K_1 \wedge K_2 \wedge \dots \wedge K_s$ is equivalent to

$$(C \vee conj(F_i)) \wedge (\neg C \vee conj(G_j)) \wedge conj(H_k) \quad .$$

If C is false, then this formula is equivalent to $conj(F_i) \wedge conj(H_k)$, i.e. $conj(F_i) \wedge conj(H_k)$ takes only false values. If C is true, then it is equivalent to $conj(G_j) \wedge conj(H_k)$, i.e. $conj(G_j) \wedge conj(H_k)$ takes only false values. Thus the disjunction

$$(conj(F_i) \wedge conj(H_k)) \vee (conj(G_j) \wedge conj(H_k)) \qquad (5)$$

also takes only false values. Now, let us, apply (the "normal") one of the logical distribution rules:

$$[L_1\text{-}L_8, MP] \vdash (A \vee B) \wedge C \leftrightarrow (A \wedge C) \vee (B \wedge C) \quad ,$$

obtaining that (5) is equivalent to

$$(conj(F_i) \vee conj(G_j)) \wedge conj(H_k) \quad , \qquad (6)$$

i.e., this formula also takes only false values. And – important note! – it does not contain the atom C at all.

Finally, by applying, again, (the strange) one of the distribution rules we can conclude that (6) is equivalent to $conj(conj(F_i \vee G_j)) \wedge conj(H_k)$, i.e., to the set of clauses $F_i \vee G_j$ and H_k (where i, j, k run over their initial ranges).

What does this achievement mean? If the set of propositional clauses $K_1, K_2, \ldots, K_s$ is inconsistent, then there is a set of clauses $F_i \vee G_j$ and H_k (where i, j, k run over their initial ranges), which is inconsistent as well, but which contains one different atom less than $K_1, K_2, \ldots, K_s$.

Now, imagine, that, in the clause form (4), we have applied the resolution rule for the atom C **in all the possible ways** (before applying, apply the permutation rule to reorder atoms moving C to right, and ¬C – to left):

$$\frac{F_i \vee C \, , \neg C \vee G_j}{F_i \vee G_j} \quad .$$

After this, apply the permutation and reduction rules to reduce identical atoms. In this way we have obtained exactly the above-mentioned inconsistent set of clauses $F_i \vee G_j$ and H_k (where i, j, k run over their initial ranges).

Thus, if some set of propositional formulas $K_1, K_2, \ldots, K_s$ is inconsistent, then the resolution rule (together with the permutation and reduction rules) allows to derive from it another inconsistent set of propositional formulas, which contains one different atom less.

By iterating this process, at the end of it, we will have an inconsistent set of propositional formulas built of a single atom B. In a clause form, there can be only one such set – the set B, ¬B. This set represents a contradiction.

Q.E.D.

As an example, let us use the resolution rule to prove that

$$B \vee C \, , C \rightarrow B, B \rightarrow D \quad \vdash \quad B \wedge D \quad .$$

Let us add $\neg(B \wedge D)$ to the premises $B \vee C \, , C \rightarrow B \, , B \rightarrow D$. We must prove that this set of 4 formulas is inconsistent. First, let us obtain clause forms:

$B \vee C$ in clause form is $B \vee C$,

$C \rightarrow B$ in clause form is $\neg C \vee B$,

$B \rightarrow D$ in clause form is $\neg B \vee D$,

$\neg(B \wedge D)$ in clause form is $\neg B \vee \neg D$.

Now, let us apply resolution to derive a contradiction from this set of 4 clauses: $B \vee C, \neg C \vee B, \neg B \vee D, \neg B \vee \neg D$:

From $B \vee \boldsymbol{C}, \neg \boldsymbol{C} \vee B$ we derive B, and have now 5 clauses:

$$B \vee C, \neg C \vee B, \neg B \vee D, \neg B \vee \neg D, B \quad .$$

From $\neg B \vee \boldsymbol{D}, \neg B \vee \neg \boldsymbol{D}$ we derive ¬B, and have now 6 clauses:

$$B \vee C, \neg C \vee B, \neg B \vee D, \neg B \vee \neg D, B, \neg B \quad .$$

We have derived a contradiction: B, ¬B. This proves that the formula $B \wedge D$ follows from $B \vee C, C \rightarrow B, B \rightarrow D$. Q.E.D.

Exercise 7.1.3. Use the resolution rule to prove the following:

a) A→B, ¬A→B ├ B.
b) (A→B)→A ├ A (Peirce's Law).
c) B→(C→D), B→C ├ B→D (Axiom L_2).
d) B→D, C→D ├ $B \vee C \rightarrow D$. (Axiom L_8).
e) $A \vee B \vee C, B \rightarrow A \vee C, A \rightarrow C$ ├ C.

For really usable general resolution algorithms for propositional logic that can be implemented on computers, see Davis–Putnam algorithm in Wikipedia.

Computational complexity

The resolution method cannot overcome the general complexity problem, mentioned at the end of Section 4.2: the problem of determining, is a propositional formula provable in the classical propositional logic, or not, belongs to the complexity class co-NP-complete. Indeed, imagine a set S of clauses of summary length n. A closer analysis shows that, in the **worst possible case**, the time required for the resolution method to derive a contradiction from S is exponential – about 2^{n^C} seconds (where C>0 is an absolute constant), see:

Ran Raz. Resolution lower bounds for the weak pigeonhole principle. *Journal of the Association for Computing Machinery* 51(2) (2004) pp. 115-138.

But in many **practical situations**, experience shows that resolution method solves its task in an acceptable time.

7.2. Resolution Method for Predicate Formulas

If we are interested only in deriving of contradictions from inconsistent sets of formulas, then we can note that a set of closed predicate formulas is inconsistent (i.e., allows deriving a contradiction in the classical logic) if and only if the conjunction of these formulas is unsatisfiable. Thus, instead of the initial set, we can analyze the set of *clause forms* of these formulas. If we derive a contradiction from (the union of) the set of clause forms, then this union is unsatisfiable, i.e., by Theorem 5.5.2, so is the initial set, and hence, the initial set is inconsistent. And conversely, if the initial set of formulas is consistent, then it is simultaneously satisfiable, i.e., so is the set of clause forms, i.e., we will be not able to derive a contradiction from it.

Note that clauses are disjunctions of **atomic** formulas (with or without negations), and, in clause forms, **no two clauses contain common variables**. Thus, clauses are completely separated, and this separation greatly simplifies processing of clauses by means of term substitution (see below).

Attention! To retain this principle working during the entire process of derivation, from now on, we must:

a) after any application of *permutation* and *reduction* rules: mark the processed clause as "processed", and replace it by the newly obtained one;

b) after any application of the *resolution* rule: in the newly obtained clause, rename all its variables, giving them names never used before. This is equivalent to applying an appropriate (invertible) substitution.

Will the resolution rule remain a universal tool for deriving contradictions also from inconsistent sets of predicate formulas (after reduction, from sets of non-quantified clauses)?

Imagine, we have derived the following two clauses (p is a unary predicate constant, 0 – an object constant):

$$p(x_1) \vee F(x_1, y_1) \quad , \quad \neg p(0) \vee G(x_2, y_2) \quad .$$

To apply the resolution rule, we must first, in $p(x_1)$, substitute 0 for x_1:

$$p(0) \vee F(0, y_1) \quad , \quad \neg p(0) \vee G(x_2, y_2) \quad .$$

Now, we can apply the resolution rule, obtaining the clause

$$F(0, y_1) \vee G(x_2, y_2) \quad .$$

And finally, to keep the meaning of the newly obtained clause separate from the meanings of all the other ones, let us rename its variables giving them names never used before, for example: $F(0, y_3) \vee G(x_3, y_4)$.

Surprisingly, this simple idea of "**unification by substitution**" is sufficient to make the resolution rule a universal tool for deriving of contradictions from inconsistent sets of predicate formulas!

Note. In fact, unification is a very general phenomenon in human and computer reasoning – it appears to be one of the main components in deductive, inductive and analogical reasoning:

John F. Sowa, Arun K. Majumdar. Analogical Reasoning. In: *Conceptual Structures for Knowledge Creation and Communication*, Proceedings of ICCS 2003, LNAI 2746, Springer-Verlag, Berlin, 2003, pp. 16-36. (available online).

In general, the **substitution rule** allows, for any formula F, and any term *t* such that *F(x/t)* is an admissible substitution, replace by *t* all the free occurrences of the variable *x.* The result of the substitution is denoted usually by *F(t)*.

Exercise 7.2.1. Derive the substitution rule, namely, show that [L_{12}, MP, Gen]: $F(x) \vdash F(t)$.

Theorem 7.2.1 (J. A. Robinson). In the classical predicate logic [L_1-L_{15}, MP, Gen], a set of predicate clauses (containing no common variables) is inconsistent if and only if the **resolution rule** (together with **substitution**, permutation and reduction rules) allows for deriving a contradiction from it.

Note. In some other texts, this fact is called "the refutation-completeness of the resolution rule".

Proof. Let us denote our set of clauses by $S=\{C_1, \dots, C_k\}$. Imagine a process during which the substitution, resolution, permutation and reduction rules are applied iteratively to the clauses C_i . If the set S is consistent, then none of such processes can lead to contradictions.

However, having in mind Herbrand's Theorem (Theorem 8.2.4), let us restrict the process in the following way: select a number n, and in the *first stage*, apply only the substitution rule – n times to each clause, substituting terms from the Hebrand's universe HU_S. And, in the *second stage*, apply only resolution, permutation and reduction rules.

Since no two clauses contain common variables, let us denote by x_{ij} the j-th variable appearing in the clause C_i , i.e., we can put C_i as $C_i(\boldsymbol{x_i})$, where $\boldsymbol{x_i}$ is the list of variables x_{ij} appearing in C_i . Then, the result of the first stage represents a set of nk *ground clauses* $C_i(\boldsymbol{t_{is}})(1 \le i \le k, 1 \le s \le n)$, where $\boldsymbol{t_{is}}$ is the s-th of n lists of terms t_{ijs} replacing the variables of C_i and taken from HU_S. Thus, from each clause C_i n ground clauses $C_i(\boldsymbol{t_{is}})(1 \le s \le n)$ are generated.

Clauses that do not contain variables, are called **ground clauses.**

The first stage is determined by the number n and a finite collection T of terms $t_{ijs}(1 \le i \le k, 1 \le s \le n)$ from HU_S (t_{ijs} is the s-th replacement for the j-th variable in the i-th clause C_i).

By Lemma 8.2.5, the set of ground clauses $C_i(\boldsymbol{t_{is}})(1 \le i \le k, 1 \le s \le n)$ cannot be satisfied simultaneously if and only if, considered as a set of propositional clauses, it cannot be satisfied simultaneously under the classical truth tables, i.e., if and only if this set of propositional clauses is inconsistent. By Theorem 7.1.1, a finite set of propositional clauses is inconsistent if and only if the resolution rule (together with permutation and reduction rules) allows for deriving a contradiction from it.

So, let us, in the second stage, apply the algorithm of Theorem 7.1.1 involving only resolution, permutation and reduction rules.

To summarize:

a) In the first stage, we select a number n and a finite collection T of terms $t_{ijs}(1 \le i \le k, 1 \le s \le n)$ from HU_S. And, by applying the substitution rule n times $(1 \le s \le n)$ to each clause C_i , we substitute each t_{ijs} for the j-th variable of C_i , obtaining *kn* ground clauses as the result.

b) In the second stage, we apply the algorithm of Theorem 7.1.1, using only resolution, permutation and reduction rules, and deriving (or not) a contradiction from the set of ground clauses obtained in the first stage.

By Herbrand's Theorem (Theorem 8.2.4), the collection T can be selected in such a way that the set of *kn* ground clauses $C_i(\boldsymbol{t_{is}})(1 \le i \le k, 1 \le s \le n)$ will not be satisfiable simultaneously if and only if the set S is not. So, let us select the collection T for the first stage of our process exactly in this way.

A set of formulas is inconsistent if and only if it cannot be satisfied simultaneously. Hence, in the second stage of the process a contradiction will derived if and only if the above collection of ground clauses cannot be satisfied simultaneously. And hence, if and only the initial set S cannot.

To summarize: the initial set S of clauses is inconsistent if and only if the resolution rule (together with substitution, permutation and reduction rules) allows for deriving a contradiction from it.

Q.E.D.

However, the “proof strategy” proposed in the above proof, is hopeless!

Indeed, to achieve our goal (deriving a contradiction, if the set S is inconsistent) we must **try out (in parallel) all the possible (finite) sets** of

terms t_{ijs} taken from the infinite set HU_S, and for each of them, try to derive a contradiction by using resolution, permutation and reduction rules. By Herbrand's Theorem (Theorem 8.2.4), in this way, if the initial set S is inconsistent, then we will succeed, indeed (will derive a contradiction). But here, there is a huge performance problem that does not destroy our theoretical considerations, but makes their result practically useless. The smart ideas #1 and #2 introduced below, will allow to restrict the substitution search space for the sets of terms t_{ijs} considerably and in many practical cases – make the task feasible.

Unifiers

Imagine, we succeed in the second stage, deriving a contradiction: $p_o(\boldsymbol{t_0}), \neg p_0(\boldsymbol{t_o})$, where p_0 is a predicate constant. By applying the resolution rule once again to these formulas we can obtain an empty formula.

Let us follow this derivation back to the ground clauses obtained in the first stage. In this way, we will mark the minimal subset G of all ground clauses involved in the derivation.

Now, imagine, we are starting the second stage with the formulas of the contradiction-involved subset G only, and the process ends up in an empty formula. This means that each occurrence of an atomic formula $p(\boldsymbol{t})$ participating in the ground clauses of G will be eliminated by some application of reduction or resolution rules:

$$\frac{F \vee p(\boldsymbol{t}) \vee p(\boldsymbol{t}) \vee H}{F \vee p(\boldsymbol{t}) \vee H} \quad ; \quad \frac{F \vee \neg p(\boldsymbol{t}) \vee \neg p(\boldsymbol{t}) \vee H}{F \vee \neg p(\boldsymbol{t}) \vee H} \quad ; \quad \frac{F \vee p(\boldsymbol{t}), \neg p(\boldsymbol{t}) \vee H}{F \vee H} \quad .$$

In the clauses of set G (subset of S), the occurrences of $p(\boldsymbol{t})$ were represented by some atomic formulas $p(\boldsymbol{x_1}, \boldsymbol{t_1}), p(\boldsymbol{x_2}, \boldsymbol{t_2}), \ldots$. After the substitutions of the first stage all these formulas became equal to $p(\boldsymbol{t})$. Thus, the useful substitutions of the first stage possess a specific property – they are **unifiers** of atomic formulas.

Smart idea #1: to derive a contradiction (if it exists), we can do with one specific kind of the substitution rule – the unification rule:

a) Take one or two clauses such that one can mark in them two positive atoms $p(\boldsymbol{x_j}, \boldsymbol{t_j}), p(\boldsymbol{x_k}, \boldsymbol{t_k})$, or two negative $\neg p(\boldsymbol{x_j}, \boldsymbol{t_j}), \neg p(\boldsymbol{x_k}, \boldsymbol{t_k})$, or positive and negative $p(\boldsymbol{x_j}, \boldsymbol{t_j}), \neg p(\boldsymbol{x_k}, \boldsymbol{t_k})$ atoms containing the same predicate constant.

b) Try to find for these clauses (or, one clause) a substitution *sb* making the marked atomic formulas $p(\boldsymbol{x_j}, \boldsymbol{t_j}), p(\boldsymbol{x_k}, \boldsymbol{t_k})$ equal.

c) If successful, perform the substitutions *sb.* This can be done safely because no two clauses of S contain common variables.

After this operation, the following situations are possible:

1) We have two identical atomic formulas $p(\boldsymbol{t}) \vee p(\boldsymbol{t})$ or $\neg p(\boldsymbol{t}) \vee \neg p(\boldsymbol{t})$ in a single clause. Then, apply the *reduction* rule.

Note. In some other texts, such a combination of unification and reduction is called *factoring*,

2) We have two opposite atomic formulas $p(\boldsymbol{t}) \vee \neg p(\boldsymbol{t})$ in a single clause. Then, drop this clause – being logically valid, it is useless when trying to derive a contradiction.

3) We have two identical atomic formulas $p(\boldsymbol{t})$ (or two $\neg p(\boldsymbol{t})$) in two different clauses. Then, nothing can be done.

4) We have two opposite atomic formulas $p(\boldsymbol{t})$ and $\neg p(\boldsymbol{t})$ in two different clauses. Then, apply the *resolution* rule.

Substitution "algebra"

Let us generalize the setting.

In general, a substitution involves a list of distinct variables x_1, ..., x_k and a list of terms t_1, ...,t_k (which may contain variables again). All occurrences of the variable x_i are replaced by the term t_i. Thus, this operation can be most naturally represented by the list of pairs { x_1/t_1, ..., x_k/t_k }. The result of application of some substitution sb to some expression F (term, formula, or a set of them), is denoted by F.sb.

For example, if F is p(x, f(y)) and sb = { x/f(z), y/z }, then F.sb is p(f(z), f(z)).

The empty list of pairs {} represents the empty substitution. Of course, F.{} is F, for any expression F.

The most important operation on substitutions is **composition**. If sb_1 and sb_2 are two substitutions, then $sb_1.sb_2$ denotes the composed substitution "apply first sb_1, and after this, apply sb_2". For example, if

$$sb_1 = \{ x/f(z), y/z \};\ sb_2 = \{ z/f(w) \},$$

then

$$sb_1.sb_2 = \{ x/f(f(w)), y/f(w), z/f(w) \}.$$

Lemma 7.2.2. Assume, the clause K is obtained from the clause K' by a chain of substitutions, permutations and reductions. This derivation can be converted into a proof starting with a single substitution followed by a chain of permutations and reductions.

Exercise 7.2.2. Prove Lemma 7.2.2.

Most general unifier (mgu)

How do behave unifiers in the substitution "algebra"? Assume, sb_1 and sb_2 are two different unifiers of the same pair of expressions F and G:

$$F.sb_1 = G.sb_1 \; ; \; F.sb_2 = G.sb_2.$$

If there would be a substitution sb' such that $sb_2 = sb_1.sb'$, then we could say that sb_1 is a **more general unifier** than sb_2 ("no less general" would be here a more appropriate term). For example, let us try to unify the first members of the following two formulas:

$$p(x_1) \vee F(x_1, y_1) \quad , \quad \neg p(f(x_2)) \vee G(x_{2,} y_2) \quad .$$

It would be natural to use the substitution $sb_1 = \{x_1 / f(z), x_2 / z\}$, obtaining

$$p(f(z)) \vee F(f(z), y_1) \quad , \quad \neg p(f(z)) \vee G(z, y_2) \quad .$$

But, in principle, one could use also the substitution $sb_2 = \{x_1 / f(f(z)), x_2 / f(z)\}$, obtaining

$$p(f(f(z))) \vee F(f(f(z)), y_1) \quad , \quad \neg p(f(f(z))) \vee G(f(z), y_2) \quad .$$

Of course, sb_1 is "better", because $sb_2 = sb_1.\{ z/f(z) \}$. Why? If our purpose was unifying $p(x_1)$ with $p(f(x_2))$, then sb_1 performs this (as well as sb_2), but it "leaves more space" for subsequent substitutions (than sb_2). Indeed, to continue after sb_1, instead of $sb_2 = sub_1.\{ z/f(z) \}$, we can choose also $sub_3 = sub_1.\{ z/g(z) \}$ etc. Thus, using a more general unifier is preferable.

So, let us call a unifier sb of a finite set **F** of (two or more) expressions a **most general unifier (mgu) of F** if and only if, for any other unifier sb' of **F**, there is a substitution sb" such that sb' = sb.sb".

Let us generalize the setting even further: let us consider a set $\boldsymbol{F}$ of expression lists $\boldsymbol{F_i}$, where $\boldsymbol{F_i} = \{F_{i1}, F_{i2}, ...\}$, and let us ask: are these expressions lists unifiable? A unifier must unify each of the following sets of expressions $\{F_{11}, F_{12}, ...\}; \{F_{21}, F_{22}, ...\}; ...$

Lemma 7.2.3. There is an algorithm allowing to detect, is the given set of expression lists unifiable, or not, and, if they are, constructing an mgu.

Proof. The proof below presents the idea of the original Robinson's 1965 algorithm.

Let us call *symbols* all object constants, variables, function and predicate constants appearing in the lists.

Algorithm **MGU-R** (parameter **F**: set of expression lists; returns false or mgu)

Let us parse the lists $\boldsymbol{F}_i$ in parallel, symbol by symbol, from left to right. If all lists are identical (as sequences of symbols), then they are unifiable, and their mgu is the empty substitution {}. Return {}.

Else, the empty substitution *sb={}* will be our starting point. In the process of parsing, we will add to sb a list of new susbtitutions (building, in the result, an mgu of the list).

If not all of the lists are identical, then, when parsing, we will arrive at the leftmost difference.

The following cases are possible:

a) One of the lists ends before an other one. Such lists are not unifiable. Return *false*.

b) We have arrived at two different constants c and c', or at a constant c and a function f, or at two different functions/predicates f and f'. Then the lists are not unifiable. No substitution will be able to change this difference. Return *false*.

c) We have arrived at a set of variables **x** (each in its list) and a set of equal constants c. Then any unifier of **F** must contain the substitutions **x**/c. Let us perform these substitutions all over in **F**. And let us append **x**/c to *sb*. The number of distinct variables in **F** is decreased at least by one, and F does not contain the variables affected by substitutions.

d) We have arrived at a set of variables only. Any unifier of **F** must replace these variables by identical terms. But the most general way to unify these variables is replacing them by a single variable, for example, by a new variable z that does not appear in **F**. Let us perform these substitutions all over in **F**. And let us append them to *sb*. The number of distinct variables in **F** is decreased at least by one, and **F** does not contain the variables affected by substitutions.

e) We have arrived at a function/predicate f, followed by one or more different argument lists – lists of terms: $f(\boldsymbol{t}_j)$, or $f(t_{j1},...,t_{kk})$. Then, any unifier of **F** must unify the set of lists $\{\boldsymbol{t}_1, \boldsymbol{t}_2, ...\}$. To do this, let us call recursively our algorithm **MGU-R** (this can be done safely because the set $\{\boldsymbol{t}_1, \boldsymbol{t}_2, ...\}$

does not contain the variables affected by the previous substitutions included in *sb*). If it returns *false*, do the same.

If it returns mgu – a substitution *sb'*, then let us perform *sb'* all over in **F** and append *sb'* to *sb*. The number of distinct variables in **F** is decreased at least by one, and **F** does not contain the variables affected by substitutions.

d) We have arrived at a set of variables **x** and a function/predicate f, followed by one or more different argument lists – lists of terms: $f(\boldsymbol{t}_j)$. If one of these lists, say $\boldsymbol{t}_j$, contains a variable x from the set **x**, then no substitution will be able to unify x and $f(\boldsymbol{t}_j)$. In this case, **F** is not unifiable. Return *false*.

e) But if none of the lists $\boldsymbol{t}_j$ contain the variables of **x**, then any unifier of **F** must, first of all, unify the set of lists $\{\boldsymbol{t}_1, \boldsymbol{t}_2, ...\}$, obtaining the unified list $\boldsymbol{t}$, and replace all the variables of **x** by $f(\boldsymbol{t})$. To do this, let us call recursively our algorithm **MGU-R** (this can be done safely because $\{\boldsymbol{t}_1, \boldsymbol{t}_2, ...\}$ does not contain the variables affected by the previous substitutions included in *sb*). If it returns *false*, do the same.

If it returns mgu – a substitution *sb'*, then let us perform *sb'* all over in **F** and, after this, let us replace all the variables of **x** by $f(\boldsymbol{t})$. Also, append *sb'* and $\boldsymbol{x}/f(\boldsymbol{t})$ to *sb*. The number of distinct variables in **F** is decreased at least by one, and **F** does not contain the variables affected by substitutions.

Having resolved the first difference, let us start the process from the beginning, parsing the modified **F**. Of course, if the lists of **F** are still not identical, then, by parsing, we will stop at the next difference following *after* the place where the first difference occurred.

Let us try repeating these parsing steps until no more differences are found, i.e., until all the modified lists of **F** are identical. If the process will not exit by returning *false*, then this will happen inevitably because each modification decreases the number of distinct variables in **F**. If it happens, return the substitution *sb*.

End of algorithm **MGU-R**.

Exercise 7.2.3. a) Verify that the above resulting substitution *sb* is an mgu of the set **F**. b) Verify that any mgu of **F** can be obtained from any other one by renaming variables.

Q.E.D.

The above proof presents the idea of the original Robinson's 1965 algorithm. For faster algorithms invented later see Unification (computer science) in Wikipedia.

Let us return to our main task.

Smart idea #2: to derive contradictions, we can do with an even more specific kind of the unification rule – the mgu-rule.

To formulate this rule, we can simply repeat the above formulation of the unification rule – replacing "substitutions *sb"* by *mgu.* The analysis of four above-mentioned situations with application of reduction and resolution rules remains the same.

Lemma 7.2.4. Any proof $K_1, K_2, \ldots, K_s \vdash$ K (all K-s are clauses, they do not contain common variables), where only permutation, reduction, **substitution** and resolution rules are used, can be converted into a proof $K_1, K_2, \ldots, K_s \vdash$ K' such that:

a) in the proof, only permutation, reduction, **mgu** and resolution rules are used;

b) K can be obtained from K' by a single substitution followed by a chain of permutations and reductions.

Proof. Induction by the number n of resolutions applied in the proof $K_1, K_2, \ldots, K_s \vdash$ K.

Induction base: n=0, i.e., no resolutions applied in the proof $K_1, K_2, \ldots, K_s \vdash$ K. Then K is obtained from some K_i by a chain of permutations, reductions and substitutions. Let us build an "empty" proof $K_1, K_2, \ldots, K_s \vdash K_i$. And let us compose all the substitutions used in the initial proof into a single substitution converting K_i into K.

Induction step. Assume, we have a proof $K_1, K_2, \ldots, K_s \vdash$ K, containing n+1 resolutions. Imagine the last resolution in this proof (C is an atomic formula):

$$\frac{F \vee C, \neg C \vee G}{F' \vee G'} \qquad (*).$$

The formula $F' \vee G'$ has been obtained $(F \vee G).sb_0$ where the substitution sb_0 is uniquely renaming all variables of $F \vee G$. In the proof, the step (*) is followed by a chain of permutations, reductions and substitutions. Hence, K is derived via such a chain either from the formula $F' \vee G'$, or from a formula preceding (*).

In the latter case, we can drop from the proof the step (*) and all the formulas derived from $F' \vee G'$. In this way, we obtain a proof of K containing n resolutions only, and so, can refer directly to the induction assumption.

It remains to consider the case when K has been obtained exactly from the formula $F' \vee G'$ by a chain of permutations, reductions and substitutions.

The proofs of the formulas $F \vee C, \neg C \vee G$ contain no more than n

resolutions each. Thus, by induction assumption, we can convert these proofs into permutation-reduction-**mgu-**resolution proofs of some formulas $F_1 \vee C_{1,} \neg C_2 \vee G_2$ such that:

a) $F \vee C$ is obtained from $F_1 \vee C_1$ as $(F_1 \vee C_1).sb_1$ by some substitution sb_1. Under sb_1, the atomic formula C_1 is converted into C.

b) $\neg C \vee G$ can be obtained from $\neg C_2 \vee G_2$ as $(\neg C_2 \vee G_2).sb_2$ by some substitution sb_2. Under sb_2, the atomic formula C_2 is converted into C.

Since the clauses $F_1 \vee C_1, \neg C_2 \vee G_2$ do not contain common variables, the substitutions sb_1 and sb_2 do not intersect, hence, their union $sb = sb_1 \cup sb_2$ is a unifier of C_1 and C_2. Thus, by Lemma 7.2.3, there exists an mgu_{12} of C_1 and C_2. Then, $sb = mgu_{12}.sb'$, where *sb'* is some substitutions.

Let us append to the permutation-reduction-**mgu-**resolution proofs of the clauses $F_1 \vee C_{1,} \neg C_2 \vee G_2$:

a) two applications of mgu_{12} obtaining the formulas $F_1.mgu_{12} \vee C$, $\neg C \vee G_{2.} mgu_{12}$;

b1) an application of the resolution rule obtaining the formula $F_1.mgu_{12} \vee G_{2.} mgu_{12}$;

b2) an application of some substitution *sb"* ensuring that the variables of $F_1.mgu_{12} \vee G_{2.} mgu_{12}$ are uniquely renamed.

In this way we have obtained a permutation-reduction-**mgu-**resolution proof of the formula $(F_1.mgu_{12} \vee G_{2.} mgu_{12}).sb''$.

At this point, to complete the proof, we must build a substitution converting formula $(F_1.mgu_{12} \vee G_{2.} mgu_{12}).sb''$ into formula $F' \vee G'$. Let us apply the substitution $(sb'')^{-1}.sb'.sb_0$, After $(sb'')^{-1}$ we obtain the formula $F_1.mgu_{12} \vee G_{2.} mgu_{12}$, after *sb'* (because $sb = mgu_{12}.sb'$) – the formula $(F_1 \vee G_2).sb$, i.e., $F \vee G$. And, finally, $(F \vee G).sb_0$ is $F' \vee G'$.

Since the clause *K* has been obtained from the formula $F' \vee G'$ by a chain of permutations, reductions and substitutions, let us take as *K'* the formula $(F_1.mgu_{12} \vee G_{2.} mgu_{12}).sb''$, and compose with $(sb'')^{-1}.sb'.sb_0$ all the substitutions used in the above chain, obtaining the substitution *SB*. By Lemma 7.2.2, *K* can be obtained from *K'.SB* by a chain of reductions and permutations.

Q.E.D.

Theorem 7.2.5 (J. A. Robinson). In the classical predicate logic [L_1-L_{11}, L_{12}-L_{15}, MP, Gen], a set of predicate clauses is inconsistent if and only if the **resolution rule** (together with **mgu-**, permutation and reduction rules) allows deriving a contradiction from it.

Proof. Assume, the set of clauses K_1, K_2, ... , K_s is inconsistent. Then, by Theorem 7.2.1, there are two proofs K_1, K_2, ... , K_s ├ B, K_1, K_2, ... , K_s ├ ¬B, where where only permutation, reduction, **substitution** and resolution rules are used. From clauses, these rules allow deriving only of clauses. Hence, B is an atomic formula.

By Lemma 7.2.4, both proofs can be converted into proofs K_1, K_2, ... , K_s ├ B_1, K_1, K_2, ... , K_s ├ ¬B_2 such that: a) in the proofs, only permutation, reduction, **mgu** and resolution rules are used; b_1) B can be obtained from B_1 by a single (possibly empty) substitution (permutations and reductions do not apply to atomic formulas), b_2) B can be obtained from B_2 by a single (possibly empty) substitution.

Thus, B_1 and B_2 are unifiable. Let us take their mgu, and apply it. As the result, we obtain a contradiction B', ¬B', where B' is B_1.mgu (= B_2.mgu). And we have obtained this contradiction from the clauses K_1, K_2, ... , K_s by using only permutation, reduction, **mgu-** and resolution rules.

Q.E.D.

Why is this refinement of Theorem 7.2.1 important? After Theorem 7.2.1, we were forced to try out all the possible sets of substitutions on Hebrand's universe H_S. Theorem 7.2.5 allows to restrict the substitution search space considerably. Indeed, now, we can concentrate on searching for mgu-s unifying the pairs of atomic formulas (having the same predicate constant) appearing in our clauses.

The next step – development of really working resolution strategies, for an overview, see

Stanford Introduction to Logic. An Online Course on Symbolic Logic. Stanford University, 2021, Section 12.9 "Strategies".

Computational complexity

The resolution method cannot overcome the general complexity problem, mentioned at the end of Section 4.3: by the Unsolvability Theorem, the problem of determining, is a predicate formula provable in the classical predicate logic, or not, is *not algorithmically solvable*. And indeed, all

computer programs implementing the resolution method run into loop in many situations, when the set of clauses to be processed, in fact, does not contain contradictions (and hence, they cannot be derived). But experience shows that in many **practical situations**, the resolution method solves its task, and – in an acceptable time.

For additional references, see Resolution (logic) in Wikipedia.

Horn clauses

The problem of deriving a contradiction (if it exists) from a set of clauses becomes somewhat less complicated, if in clauses

$$\neg A_1 \vee \neg A_2 \vee ... \vee \neg A_m \vee B_1 \vee B_2 \vee ... \vee B_n$$

(A_i, B_j are atomic formulas), or, alternatively,

$$A_1 \wedge A_2 \wedge ... \wedge A_m \rightarrow B_1 \vee B_2 \vee ... \vee B_n \quad ,$$

we allow only n=1 or n=0. Then, we consider

$$\neg A_1 \vee \neg A_2 \vee ... \vee \neg A_m \vee B \quad ,$$

or alternatively $A_1 \wedge A_2 \wedge ... \wedge A_m \rightarrow B$, or even (n=0): $A_1 \wedge A_2 \wedge ... \wedge A_m$.

Such clauses are called *Horn clauses* and they were named after Alfred Horn, who first noticed their significance:

A. Horn. "On sentences which are true of direct unions of algebras", Journal of Symbolic Logic, 1951, 16, 14-21.

There are formulas that cannot be reduced to Horn clauses.

The resolution rule refined for Horn clauses (the so-called *SLD resolution*) is used to implement the programming language *Prolog*.

For more details and references, see Horn clause and SLD resolution in Wikipedia.

8. Miscellaneous

8.1. Negation as Contradiction or Absurdity

The idea behind this approach is as follows: let us define ¬B (i.e., "B is false") as "B implies absurdity". So, let us add to our first order language a predicate constant f (meaning "false", or "absurdity"), and let us replace all negation expressions ¬F by F→f. Then, the three negation axioms will take the following forms:

L_9: (B→C)→((B→¬C)→¬B),

L_9': (B→C)→((B→(C→f))→(B→f)),

L_{10}: ¬B→(B→C),

L_{10}': (B→f)→(B→C),

L_{11}: $B \vee \neg B$,

L_{11}': $B \vee (B \to f)$.

After this, surprisingly, the axiom L_9' becomes derivable from L_1-L_2! Indeed,

(1)	B→C	Hypothesis assumed.
(2)	B→(C→f)	Hypothesis assumed.
(3)	B	Hypothesis assumed.
(4)	C→f	By MP, from (2) and (3)
(5)	C	By MP, from (1) and (3)
(6)	f	By MP, from (4) and (5)

Hence, by Deduction Theorem 1,

[L_1, L_2, MP]: (B→C)→((B→(C→f))→(B→f)).

Second observation. The axiom L_{10}': (B→f)→(B→C) can be replaced simply

by f→C. Indeed, if we assume f→C, then L_{10}' becomes derivable:

(1)	B→f	Hypothesis assumed.
(2)	B	Hypothesis assumed.
(3)	f	By MP, from (1) and (2)
(4)	f→C	f→C
(5)	C	By MP, from (3) and (4)

Hence, by Deduction Theorem 1, [L_1, L_2, f→C, MP]: (B→f)→(B→C).

Third observation. As we know from Section 2.4: [L_1, L_2, L_9, MP]: ¬B→(B→¬C), i.e., in the minimal logic we can prove 50% of L_{10}: "Contradiction implies that all is wrong". After our replacing negations by B→f the formula (B→f)→(B→(C→f) becomes derivable from L_1-L_2. Indeed,

(1)	B→f	Hypothesis assumed.
(2)	B	Hypothesis assumed.
(3)	f	By MP, from (1) and (2)
(4)	f→(C→f)	Axiom L_1
(5)	C→f	By MP, from (3) and (4)

Hence, by Deduction Theorem 1, [L_1, L_2, MP]: (B→f)→(B→(C→f)).

Thus, we see that L_1 (and not L_9!) is responsible for the provability of the 50% "crazy" formula ¬B→(B→¬C). Is L_1 50% as "crazy" as L_{10}? Yes! Let us compare:

L_{10}: ¬B→(B→C) states that "Contradiction implies anything".

L_1: B→(C→B) states that "If B is true, then B follows from anything".

Let us remind our "argument" in favour of L_{10} in Section 1.3: "...we do not need to know, were C "true" or not, if ¬B and B were "true" simultaneously. By assuming that "if ¬B and B were true simultaneously, then anything were true" we greatly simplify our logical apparatus."

Now, similarly: if B is (unconditionally) true, then we do not need to know, follows B from C or not. By assuming that "if B is true, then B follows from

anything" we greatly simplify our logical apparatus.

In a sense, the axiom L_9 "defines" the negation of the minimal logic, the axioms L_9 and L_{10} "define" the negation of the constructive logic, and L_9-L_{11} "define" the negation of the classical logic. Is our definition of ¬B as B→f equivalent to these "definitions"? Yes!

Theorem 8.1.1. For any formula F, let us denote by F' the formula obtained from F by replacing all sub-formulas ¬G by G→f. Then, for any formulas B_1, ..., B_n, C:

$[L_1\text{-}L_9, MP]: B_1, ..., B_n \vdash C$ if and only if $[L_1\text{-}L_8, MP]: B'_1, ..., B'_n \vdash C'$.

Proof.

1) →.

Let us consider a proof of $[L_1\text{-}L_9, MP]: B_1, ..., B_n \vdash C$. In this proof:

− let us replace each formula G by its "translation" G',

− before each instance of L_9, let us insert a proof of the corresponding instance of L'_9 in $[L_1, L_2, MP]$ (see above).

In this way we obtain a proof of $[L_1\text{-}L_8, MP]: B'_1, ..., B'_n \vdash C'$. Indeed,

a) If some formula B is an instance of L_1-L_8, then B' is an instance of the same axiom (verify!).

b) (B→D)' is B'→D', hence, if the initial proof contains a conclusion by MP from B and B→D to D, then, in the derived proof, it is converted into a conclusion by MP from B' and B'→D' to D'.

c) If the initial proof contains an instance of L_9, then the derived proof contains the corresponding instance of L'_9 preceded by its proof in $[L_1, L_2, MP]$.

Q.E.D.

2) ←.

Let us remind the above translation operation: for any formula F, we denoted by F' the formula obtained from F by replacing all sub-formulas ¬G by G→f. Now, let us introduce a kind of a converse operation – the re-translation operation: for any formula F, let us denote by F" the formula obtained from F: a) by replacing all sub-formulas G→f by ¬G, and after this, b) by replacing all the remaining f's (f means "false"!) by ¬(a→a), where a is some closed formula of the language considered.

Of course, for any formula F, (F')" is F (verify).

Note. Replacing f by a formula preceded by negation, is crucial – it will allow applying of [L_1-L_9, MP]: ¬B→(B→¬C) instead of the Axiom L_{10}: ¬B→(B→C).

Now, let us consider a proof of [L_1-L_8, MP]: B'_1, ..., B'_n ⊢ C'. In this proof, let us replace each formula G by its re-translation G". Then C' becomes C, and B'_1, ..., B'_n become B_1, ..., B_n, but what about the remaining formulas contained in the proof?

a) Instances of the axioms L_1-L_8.

L_1: B→(C→B)

If B is not f, then (B→(C→B))" is B"→(C"→B"), i.e., re-translation yields again an instance of L_1.

If B is f, then (f→(C→f))" is ¬(a→a)→¬C". This formula is provable in [L_1-L_9, MP]. Indeed,

(1)	¬(a→a)	Hypothesis assumed.
(2)	¬(a→a)→((a→a)→¬C")	[L_1-L_9, MP]: ¬B→(B→¬C)
(3)	a→a	[L_1-L_2, MP]: A→A
(4)	¬C"	By MP, from (1), (2) and (3).

Thus, re-translation of any instance of L_1 is provable in [L_1-L_9, MP].

L_2: (B→(C→D))→((B→C)→(B→D))

If C and D are not f, then re-translation yields again an instance of L_2.

If C is f, and D is not, then re-translation yields

(B"→(¬(a→a)→D"))→(¬B"→(B"→D")).

This formula is provable in [L_1-L_9, MP]. Indeed,

(1)	B"→(¬(a→a)→D")	Hypothesis assumed.
(2)	¬B"	Hypothesis assumed.
(3)	B"	Hypothesis assumed.

(4)	¬(a→a)→D"	By MP, from (1) and (3).
(5)	¬B"→(B"→¬(a→a))	[L_1,L_2, L_9, MP]: ¬B→(B→¬C)
(6)	¬(a→a)	By MP, from (2), (3) and (5).
(7)	D"	By MP, from (4) and (6).

Hence, by Deduction Theorem 1,

[L_1-L_9, MP]: (B"→(¬(a→a)→D"))→(¬B"→(B"→D")).

If D is f, and C is not, then re-translation yields

(B"→¬C")→((B"→C")→¬B").

This formula is provable in [L_1-L_9, MP]. Indeed,

(1)	B"→¬C"	Hypothesis assumed.
(2)	B"→C"	Hypothesis assumed.
(3)	¬B"	By MP, from Axiom L_9.

Hence, by Deduction Theorem 1,

[L_1-L_9, MP]: (B"→¬C")→((B"→C")→¬B").

If C and D both are f, then re-translation yields

(B"→¬¬(a→a))→(¬B"→¬B").

This formula is provable in [L_1-L_9, MP]. Indeed,

(1)	¬B"→¬B"	[L_1-L_2, MP]: A→A
(2)	(¬B"→¬B")→ (X→(¬B"→¬B"))	Axiom L_1.
(3)	X→(¬B"→¬B")	By MP, X is B"→¬¬(a→a).

Thus, re-translation of any instance of L_2 is provable in [L_1-L_9, MP].

L_3: $B \wedge C \rightarrow B$

If B is not f, then re-translation yields again an instance of L_3.

If B is f, then re-translation yields via $\neg(f \wedge C)$ the formula $\neg(\neg(a \rightarrow a) \wedge C)$. This formula is provable in [L_1-L_9, MP]. Indeed,

(1) $\neg(a \to a) \wedge C \to \neg(a \to a)$ Axiom L_3.

(2) $\neg\neg(a \to a) \to \neg(\neg(a \to a) \wedge C)$ From (1), by Contraposition Law.

(3) (a→a)→¬¬(a→a) [L_1, L_2, L_9, MP]: A→¬¬A

(4) a→a [L_1-L_2, MP]: A→A

(5) $\neg(\neg(a \to a) \wedge C)$ By MP, from (3), (4) and (2).

Thus, re-translation of any instance of L_3 is provable in [L_1-L_9, MP].

L_4: $B \wedge C \to C$

Similarly to L_3 – re-translation of any instance of L_4 is provable in [L_1-L_9, MP].

L_5: $B \to (C \to B \wedge C)$

Re-translation yields again an instance of L_5.

L_6: $B \to B \vee C$

Re-translation yields again an instance of L_6.

L_7: $C \to B \vee C$

Re-translation yields again an instance of $L_{8.}$

L_8: $(B \to D) \to ((C \to D) \to (B \vee C \to D))$

If D is not f, then re-translation yields again an instance of L_8.

If D is f, then re-translation yields $\neg B \to (\neg C \to \neg(B \vee C))$. This formula is provable in [L_1-L_9, MP].

Exercise 8.1.1. Verify that.

Thus, re-translation of any instance of L_8 is provable in [L_1-L_9, MP].

Hence, re-translations of all (i.e., L_1-L_8) axiom instances are provable in [L_1-L_9, MP]. What about applications of MP in the initial proof? If the initial proof contains a conclusion by MP from B and B→D to D, then the following situations are possible:

a) If B and D are not f, then, in the derived proof, this conclusion is converted into a conclusion by MP from B" and B"→D" to D".

b) If B is f, and D is not, then, in the derived proof, this conclusion is converted into a conclusion by MP from ¬(a→a) and ¬(a→a)→D" to D".

c) If D is f, and B is not, then, in the derived proof, this conclusion is converted into three formulas: B", ¬B", ¬(a→a). To derive ¬(a→a) from B" and ¬B", we can use MP and

$$[L_1\text{-}L_9, MP]: \neg B"\rightarrow(B"\rightarrow\neg(a\rightarrow a)).$$

d) If B and D are both f, then, in the derived proof, this conclusion is converted into three formulas: ¬(a→a), ¬¬(a→a), ¬(a→a). Simply drop the third formula from the proof.

Thus, the re-translation operation, when applied to all formulas of a proof of $[L_1\text{-}L_8, MP]: B'_1, ..., B'_n \vdash C'$, yields a sequence of formulas that are provable in $[L_1\text{-}L_9, MP]$ from hypotheses $B_1, ..., B_n$. Hence, so is C.

Q.E.D.

This completes the proof of Theorem 8.1.1.

Corollary 8.1.2. a) A formula C is provable in the minimal propositional logic $[L_1\text{-}L_9, MP]$ if and only if $[L_1\text{-}L_8, MP]: \vdash C'$.

b) A formula C is provable in the constructive propositional logic $[L_1\text{-}L_{10}, MP]$ if and only if $[L_1\text{-}L_8, f\rightarrow B, MP]: \vdash C'$.

c) A formula C is provable in the classical propositional logic $[L_1\text{-}L_{11}, MP]$ if and only if $[L_1\text{-}L_8, f\rightarrow B, L'_{11}, MP]: \vdash C'$.

Proof. a) Consider an empty set of hypotheses in Theorem 8.1.1.

b) If $[L_1\text{-}L_{10}, MP]: \vdash C$, then $[L_1\text{-}L_9, MP]: B_1, ..., B_n \vdash C$, where hypotheses are instances of the axiom L_{10}. By Theorem 8.1.1,

$$[L_1\text{-}L_8, MP]: B'_1, ..., B'_n \vdash C'.$$

As established above, $B'_1, ..., B'_n$ can be proved by using the axiom schema f→B, i.e., $[L_1\text{-}L_8, f\rightarrow B, MP]: \vdash C'$. Q.E.D.

Now, if $[L_1\text{-}L_8, f\rightarrow B, MP]: \vdash C'$, then,

c) If $[L_1\text{-}L_{11}, MP]: \vdash C$, then $[L_1\text{-}L_9, MP]: B_1, ..., B_n \vdash C$, where hypotheses are instances of the axioms L_{10} and L_{11}. Return to case (b). Q.E.D.

Corollary 8.1.3. a) A formula C is provable in the minimal predicate logic $[L_1\text{-}L_9, L_{12}\text{-}L_{15}, MP, Gen]$ if and only if $[L_1\text{-}L_8, L_{12}\text{-}L_{15}, MP, Gen]: \vdash C'$.

b) A formula C is provable in the constructive predicate logic [L_1-L_{10}, L_{12}-L_{15}, MP, Gen] if and only if [L_1-L_8, f→B, L_{12}-L_{15}, MP, Gen]: ├ C'.

c) A formula C is provable in the classical predicate logic [L_1-L_{11}, L_{12}-L_{15}, MP, Gen] if and only if [L_1-L_8, f→B, L_{11}', L_{12}-L_{15}, MP, Gen]: ├ C'.

Exercise 8.1.2. Prove Corollary 6.1.3.

8.2. Herbrand's Theorem

Attention! The principal results of this Section are valid only for the **classical logic**!

Jacques Herbrand (1908-1931) "... After leaving Göttingen, Herbrand decided on a holiday in the Alps before his intended return to France. However he was never to complete his plans for he died in a mountaineering accident in the Alps only a few days after his holiday began. His death at the age of 23 in one of the tragic losses to mathematics." (according to MacTutor History of Mathematics archive).

Herbrand proved his famous theorem in 1929:

J.Herbrand. Recherches sur la théorie de la démonstration. Ph.D. Thesis, University of Paris, 1930 (approved in April 1929).

Unlike the proof presented below, the original proof of Herbrand's Theorem does not depend on Gödel's Completeness Theorem (or Model Existence Theorem). Herbrand completed his Ph.D. thesis in 1929. In the same 1929 Gödel completed his doctoral dissertation about completeness (see Section 4.3). In fact, Herbrand's method allows proving of Gödel's Completeness Theorem, but he (Herbrand) "did not notice it". Why? See

Samuel R. Buss. On Herbrand's Theorem. "Lecture Notes in Computer Science", Vol. 960, 1995, Springer-Verlag, pp.195-209 (available online).

The flavor of Hebrand's theorem can be best presented in its simplest version. In this version, F(x) is a quantifier-free formula containing only one variable x in a predicate language, containing at least one object constant. Then, Herbrand's Theorem says:

The formula $\exists x F(x)$ is provable in the classical logic if and only if there is a finite set of constant terms t_1, ..., t_n such that the disjunction $F(t_1) \vee ... \vee F(t_n)$ is provable in the classical logic.

As we will see in the proof, Herbrand's theorem is "caused" by the simple fact that in any proof of $\exists x F(x)$ only a finite set of terms can be used.

Now, more precisely:

Let L be a predicate language, containing at least one object constant, and let F

be a quantifier-free formula.

Idea #1. The formula $p(c_1) \wedge q(c_2, f(x))$ is quantifier-free (c_1, c_2 are object constants, f – a function constant, p, q – predicate constants). In a sense, any "closed" interpretation domain for this formula must contain objects denoted by the terms c_1, c_2, $f(c_1)$, $f(c_2)$, $f(f(c_1))$, $f(f(c_2))$,...

So, let us define the so-called **Herbrand's universe** of the formula F (let us denote it by HU_F) as the minimum set of all constant terms such that:

a) If c is an object constant occurring in F, then c is in HU_F.

b) If F does not contain object constants, then one of the constants of the language L is in HU_F.

c) If terms t_1, ..., t_k are in HU_F, and f is a k-ary function constant occurring in F, then the term $f(t_1, ..., t_k)$ is in HU_F.

Exercise 8.2.1. Verify that HU_F is a non-empty finite or countable set (provide an algorithm generating all members of HU_F).

Theorem 8.2.1 (Herbrand's Theorem – the simplest case I). Let L be a predicate language, containing at least one object constant, and let F(x) be a quantifier-free formula containing only one free variable x. Then the formula $\exists x F(x)$ is **provable in the classical predicate logic** if and only if there is a finite set of terms t_1, ..., t_n from the Herbrand's universe HU_F such that the disjunction $F(t_1) \vee ... \vee F(t_n)$ is **provable in the classical predicate logic**.

Proof. A formula is provable in the classical logic if and only if it is logically valid (Gödel's Completeness Theorem).

So, let us assume the contrary – that none of the disjunctions $F(t_1) \vee ... \vee F(t_n)$ is provable in the classical logic (t_i-s are terms from HU_F). **Idea #2** – then the following theory T is consistent:

$$T = \{ \neg F(t) \mid t \text{ is a term from } HU_F \}.$$

Indeed, if T would be inconsistent, then there would be a T-proof of some formula $B \wedge \neg B$. In this proof, only a finite set of the axioms ¬F(t) would be used, i.e., for some terms t_1, ..., t_n from HU_F:

$$[L_1\text{-}L_{15}, \text{MP, Gen}]: \quad \neg F(t_1), ..., \neg F(t_n) \vdash B \wedge \neg B .$$

Hence, by Deduction Theorem 2 (it is applicable here, because F(x) contains only one free variable, and t_i-s are constant terms, i.e., every $\neg F(t_i)$ is a

closed formula):

$$[L_1\text{-}L_{15}, MP, Gen]: \vdash \neg F(t_1) \wedge ... \wedge \neg F(t_n) \rightarrow B \wedge \neg B \ ,$$

$$[L_1\text{-}L_{15}, MP, Gen]: \vdash \neg(F(t_1) \vee ... \vee \neg F(t_n)) \rightarrow B \wedge \neg B \ ,$$

and thus,

$$[L_1\text{-}L_{15}, MP, Gen]: \vdash F(t_1) \vee ... \vee F(t_n) \ .$$

This contradicts our assumption, that none of the disjunctions $F(t_1) \vee ... \vee F(t_n)$ is provable. Hence, T is a consistent theory.

Idea #3 – if T is consistent, then, by the Model Existence Theorem, there is a model J of T. In this model, all the axioms of T are true, i.e., so are all the formulas $\neg F(t)$ with t from HU_F.

Idea #4 – let us restrict the domain of the model J to those elements of it, which are interpretations of terms from HU_F, and let us restrict the entire interpretation correspondingly. Let us denote this new interpretation by J_1. Then,

a) All the formulas $\neg F(t)$ (with t from HU_F) are true in J_1. Indeed, $\neg F(t)$ contains only constant terms from HU_F (idea #1 working!), and all of them have the same interpretations in J_1 that they had in J. Thus, if $\neg F(t)$ was true in J, it remains true in J_1.

b) Hence, the formula $\forall x \neg F(x)$ is true in J_1 (because the domain of J_1 consists only of those elements, which are interpretations of terms from HU_F).

c) Hence, the formula $\exists x F(x)$ is false in J_1.

This contradicts the logical validity of $\exists x F(x)$.

Q.E.D.

Exercise 8.2.2. Repeat the above proof, proving a more general form of Herbrand's Theorem:

Theorem 8.2.2 (Herbrand's Theorem – the simplest case II). Let L be a predicate language, containing at least one object constant, and let $F(x_1, ..., x_m)$ be a quantifier-free formula containing only m free variables $x_1, ..., x_m$. The formula $\exists x_1 ... \exists x_m F(x_1, ..., x_m)$ is **provable in the classical predicate logic** if and only if there is a finite set of m-tuples $\boldsymbol{t_1}, ..., \boldsymbol{t_n}$ of terms from Herbrand's universe HU_F such that the disjunction $F(\boldsymbol{t_1}) \vee ... \vee F(\boldsymbol{t_n})$ is **provable in the classical predicate logic**.

Any formula G is logically valid if and only if ¬G is unsatisfiable. Thus, $\exists x_1 ... \exists x_m F(x_1, ..., x_m)$ is logically valid if and only if its negation

$$\forall x_1 ... \forall x_m \neg F(x_1, ..., x_m)$$

is unsatisfiable. On the other hand, $F(\boldsymbol{t_1}) \vee ... \vee F(\boldsymbol{t_n})$ is logically valid if and only if its negation $\neg F(\boldsymbol{t_1}) \wedge ... \wedge \neg F(\boldsymbol{t_n})$ is unsatisfiable. Now, let us replace F by ¬F, and we have proved

Theorem 8.2.3 (Herbrand's Theorem – a more useful alternative form). Let L be a predicate language, containing at least one object constant, and let $F(x_1, ..., x_m)$ be a quantifier-free formula containing only m free variables $x_1, ..., x_m$. The formula $\forall x_1 ... \forall x_m F(x_1, ..., x_m)$ is **unsatisfiable** if and only if there is a finite set of m-tuples $\boldsymbol{t_1}, ..., \boldsymbol{t_n}$ of terms from HU_F such that the formulas $F(\boldsymbol{t_1}), ..., F(\boldsymbol{t_n})$ **cannot be satisfied simultaneously**.

Why is this form "more useful"? Let us try applying it to sets of formulas in clause form. An interesting effect!

1) A clause is a disjunction of atomic formulas or their negations. For example, $\neg p(c_1) \vee p(c_2) \vee q(x, f(y))$, or $p(z) \vee \neg q(u, f(w))$. A *clause form* is a set of clauses, in which no two clauses contain common variables.

2) As we know from the Section 5.5, any finite set of *closed* formulas $F_1, F_2, ..., F_k$ can be reduced to a clause form, i.e., a set $S = \{C_1, ..., C_K\}$ of clauses (in a language obtained by adding a finite set of new object constants and new function constants) that cannot be satisfied simultaneously if and only if the set $F_1, F_2, ..., F_k$ cannot.

3) Formally, a set S of clauses represents a single formula – a universally quantified conjunction of all clauses from S. For the above example, it would be the formula:

$$\forall x \forall y \forall z \forall u \forall w (\neg p(c_1) \vee p(c_2) \vee q(x, f(y))) \wedge (p(z) \vee \neg q(u, f(w)))$$

So, let us apply to the set S the above form of Herbrand's Theorem (Theorem 8.2.3).

Let $x_1, ..., x_m$ be the complete list of variables appearing in S. Since no two clauses contain common variables, let us denote by x_{ij} the j-th variable appearing in the clause C_i, i.e., we can put C_i as $C_i(\boldsymbol{x_i})$.

Thus, S represents the formula $\forall x_1 ... \forall x_m D(x_1, ..., x_m)$, where D is the conjunction $C_1(\boldsymbol{x_1}) \wedge ... \wedge C_K(\boldsymbol{x_K})$ and $\boldsymbol{x_1} \cup ... \cup \boldsymbol{x_K} = \{x_1, ..., x_m\}$.

Now, let us apply Theorem 8.2.3: there exists a collection of m-tuples $\boldsymbol{t_s} (1 \le s \le n)$ of terms from Herbrand's universe HU_S such that the set

$D(\boldsymbol{t_1}), \ldots, D(\boldsymbol{t_n})$ cannot be satisfied simultaneously if and only if the set S cannot.

In fact, we have here a collection of terms $t_{ijs}(1 \le i \le K, 1 \le s \le n)$, where t_{ijs} is substituted for the j-th variable of the clause C_i in the formula $D(\boldsymbol{t_s})$. By re-grouping these terms by j as $\boldsymbol{t_{is}}(1 \le i \le K, 1 \le s \le n)$, we can rewrite $D(\boldsymbol{t_s})$ as $C_1(\boldsymbol{t_{1s}}) \wedge \ldots \wedge C_K(\boldsymbol{t_{Ks}})$. Thus, we have obtained a collection of nK variable-free clauses $C_i(\boldsymbol{t_{is}})(1 \le i \le K, 1 \le s \le n)$ that cannot be satisfied simultaneously if and only if the initial set of formulas $F_1, F_2, \ldots, F_k$ cannot.

If we take a clause, and substitute some terms from HU_S for all its variables, then we obtain a *variable-free clause*, the so-called **ground clause** of S. For example, if

$$S = \{ \ \neg p(c_1) \vee p(c_2) \vee q(x, f(y)) \ ; \ p(z) \vee \neg q(u, f(w)) \ \},$$

then the substitution $f(c_1)/x; c_2/y; c_1/z; c_2/u; f(c_2)/w$ yields the following two ground clauses:

$$\neg p(c_1) \vee p(c_2) \vee q(f(c_1), f(c_2)) \ , \ p(c_1) \vee \neg q(c_{2,} f(f(c_2))) \ .$$

Thus, we have obtained

Theorem 8.2.4 (Herbrand's Theorem – an even more useful form, author – Herbert B.Enderton?). Let L be a predicate language, and $F_1, F_2, \ldots, F_k$ be a set of of closed formulas in L. Let us obtain a clause form of this set – a set $S = \{C_1, \ldots, C_K\}$ of K clauses in the language obtained from L by adding a finite set of new object constants and new function constants. Then there exists a number n and a finite set of terms $t_{ijs}(1 \le i \le K, 1 \le s \le n)$ from the Herbrand's universe HU_S such that after substituting each t_{ijs} for the j-th variable of C_i we obtain a set of Kn **ground clauses** $C_i(\boldsymbol{t_{is}})(1 \le i \le K, 1 \le s \le n)$, that cannot be satisfied simultaneously if and only if the initial formulas $F_1, F_2, \ldots, F_k$ cannot.

Now, it remains one final step, and we will obtain Herbrand's original result – a kind of "reduction of predicate logic to propositional logic".

The above example set of ground clauses:

$$\neg p(c_1) \vee p(c_2) \vee q(f(c_1), f(c_2)) \ ,$$
$$p(c_1) \vee \neg q(c_{2,} f(f(c_2))) \ ,$$

contains 4 different atomic formulas:

$$p(c_1), p(c_2), q(f(c_1), f(c_2)), q(c_{2,} f(f(c_2))) \ .$$

Let us denote these atoms by Q_1, Q_2, Q_3, Q_4 . In this way, we have obtained two propositional clauses:

$$\neg Q_1 \vee Q_2 \vee Q_3;$$
$$Q_1 \vee \neg Q_4.$$

1. If these formulas **could not** be satisfied simultaneously under the classical truth tables, then we could not assign truth values to predicates p, q in a way making the respective ground clauses simultaneously true.

2. But, in fact, these formula **can** be satisfied under the classical truth tables – we can find a truth-value assignment making it true, for example:

Q_1=false (this makes the first disjunction true),

Q_4=false (this makes the second disjunction true).

After this, we can define the following interpretation J making all the respective ground clauses true:

D_J = { c_1, c_2, f(c_1), f(c_2), f(f(c_1), f(f(c_2), ... } (Herbrand's universe);

interpretations of predicate constants:

p(c_1)=false,

q(c_2, f(f(c_2))=false.

These assignments make both ground clauses true. All the other truth-values of p and q are irrelevant, so, we can define them in an arbitrary way (all as true, for example).

Now, the general case:

Lemma 8.2.5. A finite set of ground clauses cannot be satisfied simultaneously if and only if these clauses considered as **propositional clauses** cannot be satisfied simultaneously under the classical truth tables.

Proof. We have a set of ground clauses $C_1, C_2, \ldots, C_B$. Let us denote the atomic formulas contained in them by $Q_1, Q_2, \ldots$. In this way clauses become propositional clauses $G_1, G_2, \ldots, G_B$. In every interpretation of out predicate language, every atom Q_i obtains a definite truth value.

There are two possibilities:

a) Under the classical truth tables, the set $G_1, G_2, \ldots, G_B$ cannot be satisfied simultaneously. Then no interpretation can make all of our ground clauses $C_1, C_2, \ldots, C_B$ true.

b) The set $G_1, G_2, \ldots, G_B$ can be satisfied simultaneously. Then there is a

truth-value assignment to $Q_1, Q_2, \ldots$ making every clause G_j true. Let us define the following interpretation J making true all the ground clauses $C_1, C_2, \ldots, C_B$:

D_J = Herbrand's universe of the terms (as character strings) contained in our ground clauses.

Each Q_i is an atomic formula $p(t_1, \ldots, t_r)$, where p is a predicate constant, and t_i are terms – elements of D_J . So, let us define the interpretation of $p(t_1, \ldots, t_r)$ as true or false according to the truth-value of Q_i . These assignments are consistent because each Q_i represents a different formula. All the other truth-values of predicates do not matter and can be defined (for example) as true.

Let us define interpretations of function constants like as in the proof of the Model Existence Theorem – as the "syntactic constructor functions", i.e., if f is an k-ary function constant, and t_1, ..., t_k are constant terms, then the interpreted value f(t_1, ..., t_k) is defined simply as the character string "f(t_1, ..., t_k)" (quotation marks ignored).

And interpretations of object constants we define as these constants themselves.

The interpretation J makes all our ground clauses $C_1, C_2, \ldots, C_B$ true, indeed. So, they can be satisfied simultaneously.

Q.E.D.