**FACULTY OF COMPUTER SCIENCE AND CYBERNETICS OF THE TARAS SHEVCHENKO NATIONAL UNIVERSITY OF KYIV**
**DEPARTMENT OF THEORY AND TECHNOLOGY OF PROGRAMMING**

M. S. Nikitchenko

# COMPOSITIONAL LOGICS

**A textbook**

Kyiv, 2025

Навчальне видання

**НІКІТЧЕНКО** Микола Степанович

Рекомендовано до друку вченою радою факультету
комп'ютерних наук та кібернетики
(протокол № 14 від 20.05.2025)

Ухвалено науково-методичною комісією факультету
комп'ютерних наук та кібернетики
(Протокол № 9 від 19.05.2025)

# КОМПОЗИЦІЙНІ ЛОГІКИ

Навчальний посібник
*(Англ. мовою)*

*Київ – 2025*

UDC 510.6

Author:
M. S. Nikitchenko,

Reviewers: Dr.Sc., Prof., V. M. Tereshchenko, Ph.D., Associate Prof., O. I. Chentsov

Recommended for publication by Academic Council of the Faculty of Computer Science and Cybernetics of Taras Shevchenko National University of Kyiv (protocol №14, 20.05.2025)

Approved by the Scientific and Methodological Commission of the Faculty of Computer Science and Cybernetics (protocol №9, 19.05.2025)

COMPOSITIONAL LOGICS : a textbook / M. S. Nikitchenko. – Kyiv, 2025.– 233 p.

Compositional logics form a broad area of mathematical logic. They are in the theoretical basis of artificial intelligence, computer science, and software engineering. The purpose of this textbook is to broaden knowledge of compositional logics, their methodological basis, relations between logical systems of various types, their syntax and semantics, types of consequence and inference relations. Traditional propositional, predicate, and modal logics as well as program logics are considered as partial cases of compositional logics.

The book is for students of computer science and software engineering specialties, graduate students and teachers of computer training faculties of higher education institutions and also for specialists that develop logical systems and methods in different areas.

**CONTENTS**

**List of abbreviations and notations ..... 7**

**Introduction ..... 8**

**Chapter 1. Methodological basis of compositional logics ..... 14**
1.1. Analysis of conventional definitions of mathematical logic ..... 14
1.2. The main initial characteristics of logic ..... 20
1.3. Identifying initial concepts of logic: naïve approach ..... 25
1.4. Methodological principles and schemes ..... 28
1.5. Diagram of the main concepts of logic and their interrelations ..... 32
1.6. Levels of Activity ..... 36
1.7. Three-level scheme of logical concepts ..... 39
1.8. Methodological basis: summary ..... 43
1.9. Questions and exercises ..... 44

**Chapter 2. Constructing a hierarchy of compositional logics 46**
2.1. Abstract logic ..... 46
2.2. Propositional logic as a concretization of abstract logic 52
2.3. Predicate logic as a concretization of propositional logic 57
2.4. Modal logic as a concretization of predicate logic ..... 63
2.5. Questions and exercises ..... 66

**Chapter 3. Predicate algebras and logics ..... 68**
3.1. Algebras of quasiary predicates ..... 68
3.2. Composition-nominative logics of quasiary predicates . 72
3.3. Satisfiability problem for composition-nominative logics 80
3.4. First-order logic of partial quasiary predicates with the complement composition ..... 94
3.5. Questions and exercises ..... 98

**Chapter 4. Hierarchical nominative data ..... 100**
4.1. Different types of hierarchical nominative data ..... 100
4.2. Representation of data structures by nominative data..... 103
4.3. Algebras of nominative data ..... 107
4.4. Operational semantics of operations on nominative data 116
4.5. Questions and exercises ..... 127

**Chapter 5. Program algebras and logics ..... 128**
5.1. The main aspects of programs ..... 128
5.2. Example of formalization of a simple program ..... 132
5.3. Program algebra of quasiary mappings ..... 137
5.4. Program specification algebra of quasiary mappings .. 139
5.5. Program specification logic over hierarchical nominative data..... 142
5.6. Questions and exercises..... 146

**Chapter 6. Deductive systems for compositional logics ..... 148**
6.1. Sequent calculus for classical propositional logic ..... 148
6.2. Sequent calculus for classical predicate logic ..... 165
6.4. Sequent calculus for pure logic of quasi-ary predicates177
6.5. Sequent calculus for predicate logic over hierarchical nominative data with the complement composition.... 179
6.6. Floyd–Hoare calculus for programs with partial predicates ..... 182
6.7. Questions and exercises..... 205

**Chapter 7. Intensionality in compositional logics ..... 208**
7.1. Intensional and extensional aspects in logic..... 208
7.2. Intensionalized data..... 211
7.3. Computability over intensionalized data ..... 215
7.4. Predicate logics over intensionalized data..... 222
7.5. Questions and exercises..... 225

**Conclusions..... 227**

**References ..... 228**

**List of abbreviations and notations**

ADT — abstract data type;
ARS — abstract rewriting system;
CNL — composition-nominative logic;
CNP — composition-nominative programming;
CP — composition programming;
DA — Dijkstra algebra;
GA — Glushkov algebra;
FCL — classical quantifier-free logics;
MAS — many-sorted algebraic system;
PCL — classical propositional logic;
PCNL — propositional composition-nominative logics;
QCL — classical first-order predicate logics;
QCNL — quantifier composition-nominative logics;
RCNL — renominative composition-nominative logics.
SAA — Glushkov system of algorithmic algebras;

## Introduction

The title of this textbook refers, first of all, to the discipline “Compositional logics” for Master students of the scientific-educational program “Artificial Intelligence” at the Faculty of Computer Science and Cybernetics of the Taras Shevchenko national university of Kyiv, still, the textbook will be also useful for students in the field of “Information Technologies” who study mathematical logic.

In general, compositional logics form a broad area of mathematical logic. They are in the theoretical basis of artificial intelligence, computer science, and software engineering. They are widely used for specification and verification of software systems. But such usage is rather complicated due to the changes that have occurred in recent years in the triad "student – course content – teacher".

Let's start with students. The constant and rapid change in information technologies and new methods of obtaining information, the dominance of "clip" and utilitarian thinking, decreasing of interest in theoretical knowledge, the decline in the level of mathematical training make it difficult for students to understand the basics of mathematical logic. As for teachers, they can be roughly divided into two groups: the first includes experienced teachers with a thorough mathematical training, but a weak level of knowledge of modern information technologies and the use of mathematical logic in artificial intelligence and software development; the second includes younger teachers with knowledge of information technologies, but with insufficient mathematical training. This situation is not surprising, because in Ukraine there are practically no departments of mathematical logic that would train highly professional specialists. Finally, such a component of the triad as the content of the course also does not meet modern requirements. The fact is that the founda-

tions of mathematical logic were laid in the middle of the 20th century, when the main attention was paid to purely theoretical issues of describing axiomatic systems (proof theory) and theoretical issues of truth (model theory). The wide application of logic poses more acute problems of focusing logic on applied problems and on integrating knowledge focused on different subject areas. Although quite a few textbooks have been published that are focused on teaching mathematical logic for students in the field of "Information Technologies", they are still mainly limited in the theoretical component to teaching propositional logic, first-order logic and sometimes temporal logic, and in the practical component to considering certain applications of logic; a typical example is a textbook [Ben-Ari2022]. Such an approach is fragmentary and does not allow for the formation of a holistic understanding of logic.

Among the difficulties that students encounter when studying theoretical disciplines, and, in particular, mathematical logic, we distinguish the following:

1) insufficient methodological preparation;

2) vague understanding of general categories and professional concepts;

3) weak skills of explication and formalization of knowledge.

Let us provide some explanations for the above difficulties.

We start with methodological issues. The most important method (principle) is the ascent from the abstract to the concrete [Hegel1969]. The application of this method requires a clear understanding of the categories of abstract and concrete. Unfortunately, students often interpret these categories as imaginary (ideal) and real. Such an interpretation is also supported by dictionaries. But in our case, these terms should be understood as one-sided and the unity of the many-sided, respectively. In computer science, this method (in a limited version) is the dominant method of software development. Therefore, in particular, in educational programs of specialties in the field of "Information Technologies" the first general compe-

tence is formulated as "the ability to abstract thinking, analysis and synthesis" [ОПП2019]. Mastering this competence requires the disclosure not only of the categories of abstract and concrete, but also of the categories related to analysis and synthesis, which is currently a weak point in the educational process.

Let us now demonstrate the difficulties in defining professional concepts. Consider, for example, the definition of a formal grammar, which usually has the following form: a formal grammar is the quadruple G = (N, T, P, S), where N and T are finite alphabets of non-terminal and terminal symbols, respectively ($N \cap T = \emptyset$), P is a finite set of productions (rules), S is an initial symbol (axiom).

It is not difficult to understand that the specified quadruple G = (N, T, P, S) is not actually a complete definition of a formal grammar because nothing is said about a language generated by the grammar. The four sets given specify only certain individual parameters of a specific grammar; in this sense, they can be called extensional characteristics of a grammar. As for the general characteristics of a formal grammar, namely, the mechanism of generation and the definition of the language generated by the grammar, they are defined separately from the definition of the grammar itself. We will call such general characteristics intensional. Thus, the above definition of grammar is partial, it is extensional, and therefore not complete, because it emphasizes secondary extensional aspects, ignoring more important intensional aspects. The same remarks can be made regarding other concepts, for example, Turing machines.

Finally, let us consider the difficulties of mathematical formalization of computer science concepts. Usually, mathematical formalization is understood as set-theoretic formalization. Such formalization is dominant at the current stage of development of computer science. But how adequate is such formalization? For computer science, the most important concept is the concept of function. Usually, a function is understood as a binary relation between sets of input and output data [Bourbaki1968]. Such reduction of the multi-aspect concept of function to a binary relation (to a function graph)

discards other aspects of functions that are important when creating software systems, for example, such as decomposition and composition of functions. But even such a relational reduction is not adequate for complex functions. Let us demonstrate this by the example of the following Turing machine M = (Q, A, λ, δ, q0, F),

where:

– Q ={ q0, qf}– set of states;

– A ={1, λ} – alphabet;

– λ – empty symbol;

– δ = {q0 1→ q0 λ R, q0 λ → q0 1 R, q0 λ → qf} – set of commands;

– q0 – initial state;

– F ={ qf} – set of final states.

At first glance, it seems that this nondeterministic machine generates a function that transform any natural number to any natural number. In other words, the graph of this function is a complete binary relation on the set of natural numbers Nat. But this does not take into account the case when the Turing machine does not terminate, which means that the value of the function is undefined. Of course, one can try to construct another set-theoretic model of the function, which will specify the undefinedeness domain. But, as it is known from the theory of algorithms, the undefinedeness domain of a partially recursive function may not be recursively enumerable. Therefore, set-theoretic interpretations of functions are not adequate.

Many more examples can be given related to mastering the basics of logic and mathematical logic. Let us highlight another difficulty here: students know that there are different logics, in particular, dialectical logic, traditional logic, and mathematical logic. But it is difficult for them to say anything about their connection, and this complicates the understanding of the unity of the basic concepts of logic.

How can we try to improve the situation? We propose a three-level approach to teaching mathematical logic. This approach con-

sists in distinguishing three levels of consideration of logic: methodological (philosophical), professional (scientific), and formal (mathematical). At the philosophical level, logic is specified as dialectics (theory of development), at the second level – as the logic of specific sciences (applied logics), and at the third level – as mathematical logic.

Specialists in the field of information technology mainly concern themselves with the professional and formal level and do not pay enough attention to the methodological level. Although it should be noted that the importance of the methodological level is recognized in requirements to candidates for the degree of Doctor of Philosophy who must master "general scientific (philosophical) competencies aimed at the formation of a systematic scientific worldview, professional ethics and general cultural outlook" [КМУ2016]. It should only be added that mastering general scientific competencies is necessary not only for Doctors of Philosophy, but also for Bachelors and Masters (at least at the initial level).

In this textbook a new approach to mathematical logic is given. Its main characteristics are as follows:

- Basic notions of logic are considered at three interrelated levels: methodological (philosophical), professional (scientific), and formal (mathematical); therefore, three types of notions are specified: philosophical categories, scientific notions, and mathematical notions respectively;
- The initial categories are World, Subject, Object, Activity, Truth, Logic, Science, Value (a name of category begins with a capital letter);
- Within a proposed system of categories, the category of Activity is the central one, Logic is derived from this category and is defined as the Science of Truth of Activity;
- Mathematical logic is specified at the mathematical level; its main notions are projections of corresponding categories to the scientific level with their further formalizations.
- Compositional logics form a special class of mathematical logic.

The structure of the textbook is as follows:

Chapter 1 “Methodological basis of compositional logics” describes the main principles and schemes of the proposed approach, three-level diagram and levels of activities.

Chapter 2 is devoted to successive development of hierarchy of compositional logics starting from abstract logic via propositional and predicate logics up to modal logic.

Chapter 3 specifies predicate algebras, especially algebras of quasiary predicates, which adequately formalize semantics of predicate languages. Various types of such algebras and problems of formulas satisfiability are considered. For algebras of partial predicates, the complement composition is introduced and their properties are studied.

Chapter 4 investigates various classes of nominative data. Such data may be treated as an universal data structure capable to model data structure used for subject domain modelling.

Chapter 5 is oriented on studying algebras and logics used to specify properties of software systems.

Chapter 6 describes deductive systems, especially sequent calculi, for the logics described earlier.

Chapter 7 discussing a new topic of intensionality in compositional logics. Contrary to extensional approach to formalization of logics, intensional approach emphasizes importance of general (universal) aspects in logic.

The book will be useful for students of computer science and software engineering specialties, graduate students and teachers of computer training faculties of higher education institutions and also for specialists that develop logical systems and methods in different areas.

## Chapter 1. Methodological basis of compositional logics

### *1.1. Analysis of conventional definitions of mathematical logic*

The term 'compositional logic' consists of the adjective 'compositional' and the noun 'logic', thus, compositional logic is a special case of logic in general and mathematical logic in particular. Therefore, the initial step of defining compositional logics consists of specifying logic and mathematical logic.

Considering the definitions of logic, we find strange situation: though logic has a rather long history but still there no commonly excepted its definition; actually, there are a number of different definitions.

For example, a famous logician Jaakko J. Hintikka defines logic as "the study of correct reasoning, especially as it involves the drawing of inferences". This definition specifies reasoning as a subject matter of logic but the definition is too general and therefore some questions are not clear:

- what is correct reasoning?
- how inferences are related to reasoning?
- why reasoning involves drawing of inferences?

The author of a well-accepted handbook on logic, Elliott Mendelson wrote that "one of the popular definitions of logic is that it is the analysis of methods of reasoning. In studying these methods, logic is interested in the form rather than the content of the argument" [Mendelson1966].

Let us try to understand meaning of logic by etymological analysis.

Dictionaries give the following definitions:

- The noun logos derives from the Greek verb legein, meaning 'to say' something significant. Logos developed a wide variety of senses, including 'description', 'theory' (sometimes as opposed to 'fact'), 'explanation', 'reason', 'reasoning power', 'principle', 'ratio', 'prose'

(https://www.rep.routledge.com/articles/thematic/logos/v-1 )

- Ancient Greek:
  λόγος, romanized: lógos, lit. 'word, discourse, or reason' is related to Ancient Greek: λέγω, romanized: légō, lit. 'I say' which is cognate with Latin: lex, lit. 'law'. The word derives from a Proto-Indo-European root, *leǵ-, which can have the meanings "I put in order, arrange, gather, choose, count, reckon, discern, say, speak". In modern usage, it typically connotes the verbs "account", "measure", "reason" or "discourse".[1][2] It is occasionally used in other contexts, such as for "ratio" in mathematics (https://en.wikipedia.org/wiki/Logos )
- Logos, 1580s, "the divine Word, second person of the Christian Trinity," from Greek logos "word, speech, statement, discourse," also "a computation, account," also "reason, judgment, understanding," from PIE *log-o-, suffixed form of root *leg- (1) "to collect, gather," with derivatives meaning "to speak," on notion of "to pick out words." The Greek word was used by Neo-Platonists in metaphysical and theological senses involving notions of both "reason" and "word" and subsequently picked up by New Testament writers (https://www.etymonline.com/word/Logos )
- From Ancient Greek λόγος (lógos, “speech, oration, discourse, quote, story, study, ratio, word, calculation, reason”) (https://en.wiktionary.org/wiki/logos)

We see that there are dozens of meanings of the word ‘logic’. Also, there are a lot of synonyms:
word, symbols, emblems, badges, trademarks, logotypes, world spirit, label, son, tokens, brands, anima mundi, logotype, oversoul, universal ego, world soul, signposts, world principle. infinite spirit, signs, tags, trade names, universal life force, cues, guideposts, supreme soul, world-self, symbol, tickets, brand, emblem, marks, crests, markers, finger posts, indications, nous, registered trademarks, signals (https://www.powerthesaurus.org/logos/synonyms )

Let us also look at the antonyms: copyright, absolute silence, carelessness, complete silence, gibberish, silence, stupidity, written statement (https://www.powerthesaurus.org/logos/antonyms)

From the above given definitions, the main meanings of the word ‘logos’ can be listed as follows:

- A word; reason; speech
- The divine word of God; the second person in the Trinity (incarnate in Jesus), synonyms: son, word, logos
- The divine Word; Christ
- World spirit (Logos, World Reason, anima mundi, infinite spirit, oversoul, universal ego, universal life force, world principle, world soul, world-self) (https://www.thesaurus.com/browse/world spirit)

We see that linguistic and etymological considerations do not help much in understanding of the meanings of the word ‘logos’ that is relevant to compositional logic.

But maybe definitions of logic can help? The following definitions of logic are given in dictionaries:

- The branch of philosophy that analyzes inference
- Reasoned and reasonable judgment ("it made a certain kind of logic")
- The principles that guide reasoning within a given field or situation ("economic logic requires it", "by the logic of war")
- The system of operations performed by a computer that underlies the machine's representation of logical operations
- A system of reasoning (synonyms: logical system, system of logic)
- A treatise on logic; as, Mill's Logic.

(https://www.powerthesaurus.org/logic/definitions)

The situation becomes vague, the term ‘logic’ has almost one thousand of synonyms and dozens of antonyms.

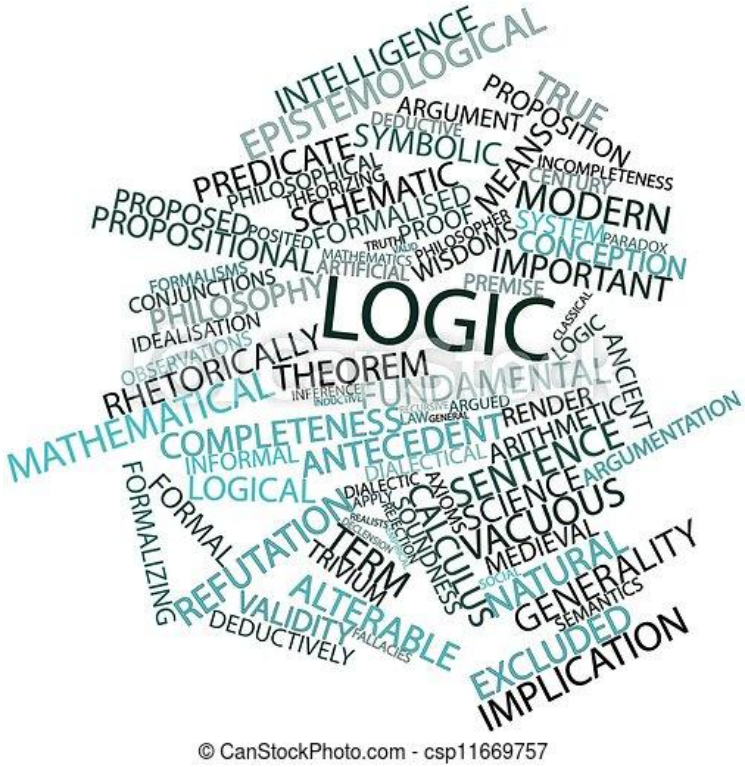
Synonyms for ‘logic’
(https://www.powerthesaurus.org/logic/synonyms):

reason
reasoning
philosophy
sense
rationality
intelligence
common sense
thinking
wisdom
argument
ratiocination
good sense
reasons
coherence
judgement
argumentation
discernment
logical
logical system
soundness
dialectic
deduction
line of reasoning
intellect
sanity
case
dialectics
logicality
horse sense
inference
perception
shrewdness
gumption
syllogism
analysis
logical thought
rationalness
reasonableness
thought
induction
ontology
oral skills
sound judgment
aesthetics
line of thought
method
rationalism
prudence
rhyme or reason
sagacity
system
acumen
contention
denotation
logical basis
logically
metaphysics
nature of the beast
sharpness
syllogistics
astuteness
consistent
grounds
senses
theory
thesis
cleverness
discrimination
explanation
generalization
justification
order
reckoning
sensibleness
brains
cogitation
logics
practicality
rationales
rumination
savvy
system of logic
theorization
understanding
wit
approach
assertion
brain
cases
claim
cognition
defense
design
exposition
logicalness
organization
perspicacity

Antonyms for logic:
(https://www.powerthesaurus.org/logic/antonyms):

foolishness
folly
play pool
madness
silliness
craziness
dullness
ignorance
incompetence
muddle
stupidity
abracadabra
absurdity
accusation
addled
apple sauce
applesauce
awkwardness
bad judgment
befuddledness
befuddlement
bemusement
bewilderedness
bewilderment
blame
bumbling
challenge
charge
chumpishness
claptrap
clumsiness
complaint
confoundedness
confoundedness of ideas
confusion of thought
contest
crookedness
daze
deceit
deception
denial
dimness
dimwittedness

Finally, let us look at the abstract word cloud for logic with related tags and terms:





**csp11669757:** logic, cloud, word, words, tag, tags, abstract, terms, keywords, term, keyword, illustration, typography, concept, refutation, theorizing, epistemological, mathematical, proposed, axioms, natural, Idealisation, time, predominated, completeness, formalizing, inductive, fundamental, philosophy, classical, modern, dialectic, valid, proposition, trivium, ancient, argument, informal, declension, theorem, inference, century, science, intelligence, formalized, real-

ists, fallacies, true, validity, paradox, recursive, schematic, premise, soundness, Quantum, empirical, rule, incompleteness, calculus, law, important, means, placeholders, rhetoric, implication, truth, apply, conception, structure, stock illustration, royalty free illustrations, stock clip art icon, stock clipart icons, logo, line art, pictures, graphic, graphics, drawing, drawings, artwork.

(https://www.canstockphoto.com/logic-11669757.html)

The given definitions and examples do not help much in understanding logic.

Summarizing, the following conclusions can be formulated:

- The absence of a generally accepted definition of logic.
- The extremely wide range of uses of logic: from the world spirit to economics, computers, arguments, quanta, etc. Billions of references to 'logic' in internet.
- The possibility of having a wide variety of logics: from very general to highly specialized.

### *1.2. The main initial characteristics of logic*

Despite the absence of the acceptable definition of logic, the conducted analysis leads to one very important characteristic of logic: the area of applicability of logic is *not restricted*, is *unbounded*, *unlimited*.

Let us give a more detailed explanation of this unboundedness scope of application of logic. For this case we will use the following terms: *universal*, *infinite*, and *absolute*.

The dictionary definitions of *universal*:

– including or covering all or a whole collectively or distributively; without limit or exception

*especially* **:** available equitably to all members of a society

*universal* health coverage

– present or occurring everywhere

– existent or operative everywhere or under all conditions
*universal* cultural patterns
– embracing a major part or the greatest portion (as of humankind)
a *universal* state
*universal* practices
– comprehensively broad and versatile
a *universal* genius
– affirming or denying something of all members of a class or of all values of a variable
– denoting every member of a class
a *universal* term
– adapted or adjustable to meet varied requirements (as of use, shape, or size)
a *universal* gear cutter
a *universal* remote control
(https://www.merriam-webster.com/dictionary/universal)

The dictionary definitions of *infinite*:

– without limits; extremely large or great
– being or seeming to be without limits

The dictionary definitions of *absolute*:

– without limit, very great, or to the largest degree possible.

These terms are similar in the meaning concerning their being without limits, but we order them with respect to their power:

– universal
– infinite
– absolute.

Such ordering is clear from the following definition of the Absolute:

“the **Absolute** is the concept of an unconditional reality which transcends limited, conditional, everyday existence. It is often used as an alternate term for a "God" or "the Divine", especially, but by no means exclusively, by those who feel that the term "God" lends

itself too easily to anthropomorphic presumptions. The concept of The Absolute may or may not (depending on one's specific doctrine) possess discrete will, intelligence, awareness or even a personal nature. It is sometimes conceived of as the source through which all being emanates. It contrasts with finite things, considered individually, and known collectively as the relative. As such, the word "Absolute" signifies a negative concept: non–relative, non– comparative, or without relation to anything else. This is reflected in its Latin origin *absolūtus* which means "loosened from" or "unattached.“ https://psychology.fandom.com/wiki/Absolute_(philosophy)

In this textbook we will primarily use terms ‘infinite’ and ‘universal’.

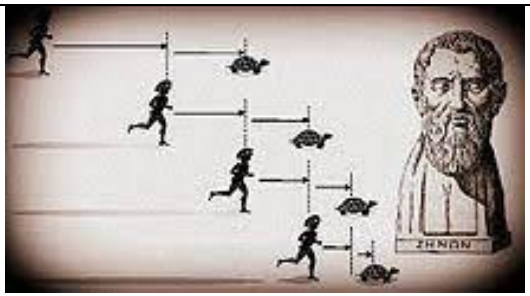
Let us analyze the ‘infinite’ in more detail.

We interpret the ‘infinite’ (universal, absolute) as unlimited. It turns out that such an infinite is contradictory.

Indeed, when we try to give some definitions of the infinite, we make it finite. This is because a *definition* is a setting of a limit, a limitation (*definition* from Latin *dēfīnītiōn-, dēfīnītiō* "fixing of a boundary, precise description," from *dēfīnīre* "to mark the limits of, determine).

But the infinite is not finite, thus a *contradiction* is obtained.

Such contradictions were known from the ancient times. It is enough to recall Zeno's aporias (paradoxes): Achilles and the tortoise, the arrow, and other aporias concerning time and space ( https://en.wikipedia.org/wiki/Zeno%27s_paradoxes).




Achilles and the tortoise

The arrow

The absolute is contradictory. Therefore, its logic will be the logic of the contradictions.

This is a very complex logic. The most developed version is *Hegel's dialectic* (not to be confused with the so-called materialist dialectic of Marx-Lenin, which is a primitivization of Hegel's dialectic).

It turns out that the concepts of finite and infinite are inextricably linked: that is, in the finite there is the infinite, and in the infinite there is the finite (the connection of part and whole).

Thus, when defining logic, we fall into a vicious circle: the infinite cannot be studied deeply without the finite, and the finite without the infinite.

A simple example: it is impossible to deeply understand the processes on Earth without taking into account the processes in the Solar System, and the Solar System without the Milky Way Galaxy, and so on up to the Universe.

*Conclusion:* The logic of the infinite must be a *contradictory unity of the infinite and the finite* (of the absolute and the relative). Such a unity is very complex, therefore there are no simple methods for constructing and studying logic of the infinite.

This situation is presented in the following picture (from Google images):

One chair represents the infinite, another represents the finite, but they should be taken in their unity.

Now let us formulate one more important characteristic: *logic is a science*. From this follows that logic has should have the main characteristic of science: objectivity, verifiability, necessity etc.

Combining the considered characteristics of logic, we obtain its preliminary definition:

***logic is a science of the infinite taken in the unity with the finite.***

We will also characterize logic as *universal science*.

Summarizing, we have formulated the preliminary (initial) characteristic of logic but have not identify its main concepts (notions) and its subject matter.

We first try to do this in a simple (naïve) manner.

### *1.3. Identifying initial concepts of logic: naïve approach*

The Merriam-Webster dictionary defines ‘naïve’ as marked by unaffected simplicity, deficient in wisdom or informed judgment, primitive.

As an example, let us recall the famous story about a naïve little boy who asked Euclid "What will I get from geometry?". Angry Euclid gave him an obol (a small coin) and demanded that the boy no longer come to his geometry school.

This example raises at least two questions concerning geometry: 'Is geometry good?' and 'Is it a theoretical or a practical discipline?'.

Answering these questions with respect to logic, we adopt that it is good (goodness) and is a science. The modern term for ‘goodness’ is ‘value’ thus it reflects an axiological aspect of logic.

Now the following two questions arise: ‘Logic is good for what?’ and ‘What are the main characteristics of logic as a science?’.

A naïve answer for the first question is ‘Logic is good for life’. This answer is too general and has many clarifications. Therefore, it is necessary to indicate the main aspect of our life. And this aspect is activity. One can find many confirmations of this thesis in life, culture, science, philosophy, and religion. For example [Proverbs],

- Happiness is activity (Aristotle);
- To live well is to work well, to show a good activity (Thomas Aquinas);
- All growth depends upon activity (Calvin Coolidge);
- Philosophy is not a theory but an activity (Ludwig Wittgenstein);
- The pursuit of truth and beauty is a sphere of activity (Albert Einstein).

Source: https://proverbicals.com/activity-proverbs

Hegel stated that “the true being of man is his action” [Filoni2022].

So, the answer to the first question is that logic is a science of activity.

As to the second question about the main characteristics of logic as a science, it is necessary to delimit characteristics common to all sciences (objectivity, necessity, verifiability, etc.), from characteristics specific to logic, which distinguish it from traditional sciences, such as mechanics, chemistry, biology, etc. The indicated sciences have their own, clearly defined areas, but logic, unlike them, is applicable to all areas. That is, its exclusivity is the boundless of logic, its universality, absoluteness.

Since logic is a science of activity, it is necessary to indicate such a characteristic of activity, which makes logic exceptional. Such a feature of activity, which is of a universal character, can be described in different ways, for example, correct activity, necessary activity, effective activity, etc. Generalizing such characteristics, we obtain that the required characteristic of activity is its truth. Truth is an exceptional word. Through ages scientists, philosophers, religious figures, and even ordinary people strive for truth.

So, the *subject matter of* ***logic*** **is the** ***truth of activity*****.**

Finally, it should be said that activity takes place in the *world*.

It means that we identify the following main notions (concepts) of Logic:

- Value (goodness)
- Science
- Universal science
- Truth
- Activity
- World

It seems that these concepts initially are poorly connected to each other, but now they need to be somehow organized. An analysis of these concepts shows that some of them are more general and others are more specific.

Let us start with concepts that can be interpreted as more general than logic: universal science is more general than logic, science is

more general than universal science, and value is more general than science because science can be considered as one of the values.

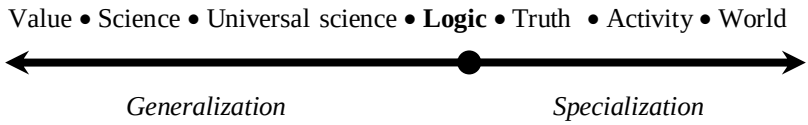
Now let's continue our consideration with concepts that are more special than logic: logic is a special concept in truth (science of truth), truth is a special characteristic of activity, in its turn, activity is a special moment of the world.

Constructing this hierarchy, we rely on dictionary definitions of the terms generalization and specialization, which are considered as opposites:

- Generalization: a general statement or concept obtained by inference from specific cases. (https://languages.oup.com/google-dictionary-en)
- Specialization: the limiting of one's study or work to one particular area, or a particular area of knowledge https://dictionary.cambridge.org/dictionary/english/specialization

Of course, this is a naïve ordering and other ordering can be proposed, but the proposed ordering opens the way for further explications.

The constructed hierarchy can be represented by the following diagram:




The possibilities of the naïve approach have actually been exhausted, now a transition to methodologically justified further development of the basic concepts of logic is required. We will first

describe the main methodological principles and schemes, selected by the orientation of our work to mathematical logic.

### *1.4. Methodological principles and schemes*

By a dictionary definition, *principle* is a comprehensive and fundamental law, doctrine, or assumption (https://www.merriam-webster.com/dictionary/principle).

We start with principles related to ontology, gnoseology, and teleology:

- the principle of universal connection;
- the principle of the ascent from the abstract to the concrete;
- the principle of orientation on mathematical logic.

Let us give short explanations of these principles.

The general *principle of universal connection* can be found in different formulation in philosophy, science, and logic (see, e.g. [Spirkin1983]). The principle states that nothing in the world stands by itself, in isolation; every object, thing, process is connected with all others uniting them in a single whole. As a consequence, all concepts of logic are interrelated and affect each other. This principle is usually treated as an ontological one.

The general *principle of the ascent from the abstract to the concrete* was explicitly formulated and used by Hegel [Hegel1969] as the main method of cognition of reality. Here abstract means one-sided, and concrete means UNITY of many sides. As a consequence, the main concepts of logic can be attributed to some levels of abstraction and concretization. This principle is usually treated as a gnoseological one.

The specific *principle of orientation on mathematical logic* means that among dozens of categories we take into consideration only those which are related to mathematical logic, its theory and its practice. This principle can be regarded as teleological one because it formulates the aims of our constructions.

The above-described principles outline general characteristics of the approach and should be supported by more specific schemes.

By a dictionary definition, *scheme* is a plan for doing or organizing something (https://dictionary.cambridge.org/dictionary/english/scheme).

Many schemes can be found in the literature. Here we are oriented on Hegel's schemes [Hegel1969] choosing those that are used in our constructions:

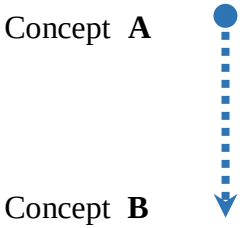
- Sublation
- Negation of negation
- Dialectical triad
- Dialectical tetrad
- Double dialectical tetrad

Georg Wilhelm Friedrich Hegel - Wikipedia

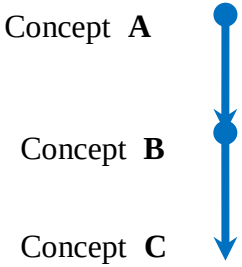
These schemes can be shortly described in the following way.

**Sublation (SB).** This is one of the main categories in Hegel's system of logic. Intuitively, sublation means to negate something

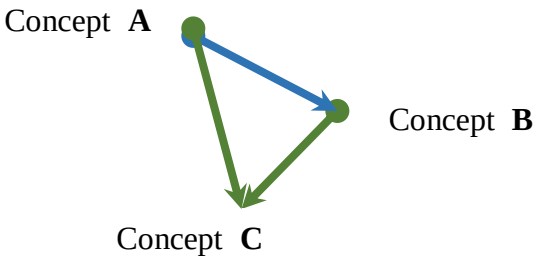
but to preserve it in some other form. Primitive examples: cream is sublated milk, butter is sublated cream.



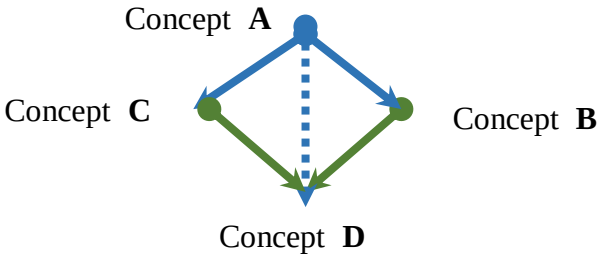

**Negation of negation (NN)**: something being negated leads to the opposite which being negated gives the new preserving the main characteristics of the initial something. Primitive example: an apple tree is negated by its seeds which in their turn are negated by a new apple tree.



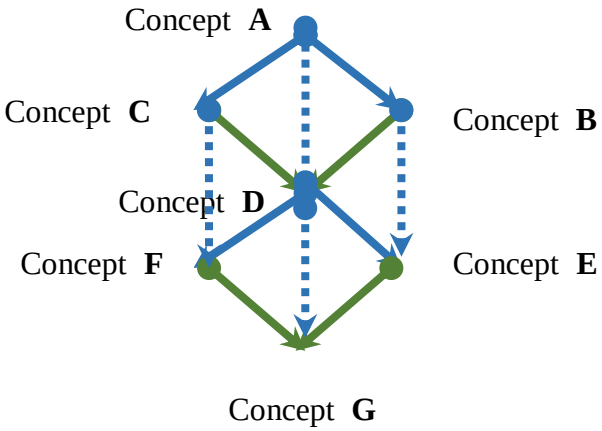

**Dialectical triad (DTR)**: something (thesis) leads to antithesis and both are united in synthesis. A primitive example: the mother-in-law is thesis, the son-in-law is antithesis, and the mother's daughter who is the wife of the son-in-law is synthesis.




**Dialectical tetrad (DTT)**: the whole is spitted into two opposites which unite in the concrete. A primitive example: a proposition is a union of syntax and semantics. This scheme reflects scientific method of cognition: we start with *syncretic* treatment of object, then analyze it, and finally make a synthesis (three levels: syncretic, analytic, and synthetic). Transition from the whole to the concrete can be regarded as sublation.




**Double dialectical tetrad (DDT)**: The concrete of the first dialectical tetrad is considered as the whole which by the second tetrad leads to a new concrete. The corresponding opposites of the two tetrads are related by sublation.




### *1.5. Diagram of the main concepts of logic and their interrelations*

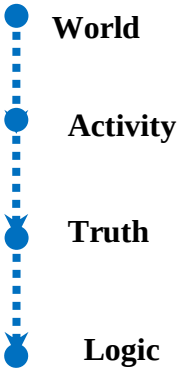
Questions concerning the relations logic–science and logic–value are beyond the scope of this book, so we focus on the relations logic–truth–activity–world.

For this purpose, a diagram of the relationships between these concepts will be constructed.

By a dictionary definition, *diagram* is a simplified drawing showing the appearance, structure, or workings of something; a schematic representation.

Previously, relationships in a chain of concepts logic–truth–activity–world were interpreted as relationships of generalization/specialization. Now, these relationships can be described more clearly, namely, it can be asserted that the most important thing in the world is activity, and in activity the most important thing is its truth, and for truth the most important thing is its laws, and the laws of truth is logic. This means that the specified concepts are connect-

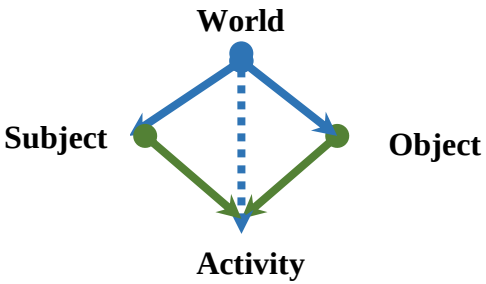
ed by the relation of sublation. Let us represent the obtained relations in the following diagram (we use capital letters to refer to categories of logic):




The constructed diagram specifies different levels of abstraction, but it does not indicate the concrete as a unity of the diverse. This means that truth and activity must be presented concretely. But how to start: with truth or activity? Here we must take into account that we are developing logic, but in order to understand logic, we must know what truth is, and since truth is sublated activity, we must start with activity.

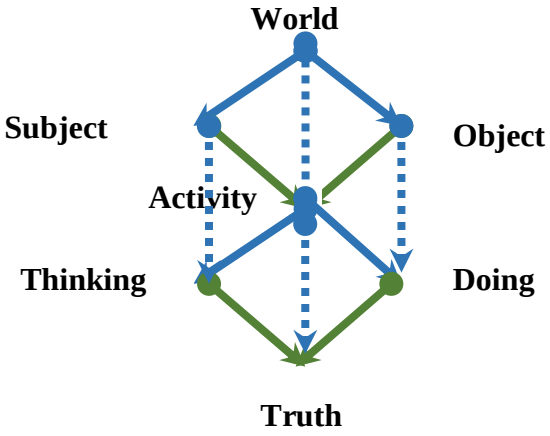
By a dictionary definition, *activity* is a process of active interaction between a *subject* and an *object*. A *subject* is someone who performs an action, or from whom a certain activity is directed. And accordingly, this activity is directed to an object. An *object* is that part of objective reality that the subject has already involved in the framework of its activity (https://uk.wikipedia.org/wiki/%D0%A1%D1%83%D0%B1'%D1%94%D0%BA%D1%82_(%D1%84%D1%96%D0%BB%D0%BE%D1%81%D0%BE%D1%84%D1%96%D1%8F).

The indicated characteristics allow us to apply the diagram of the dialectical tetrad to construct activity as the concrete:



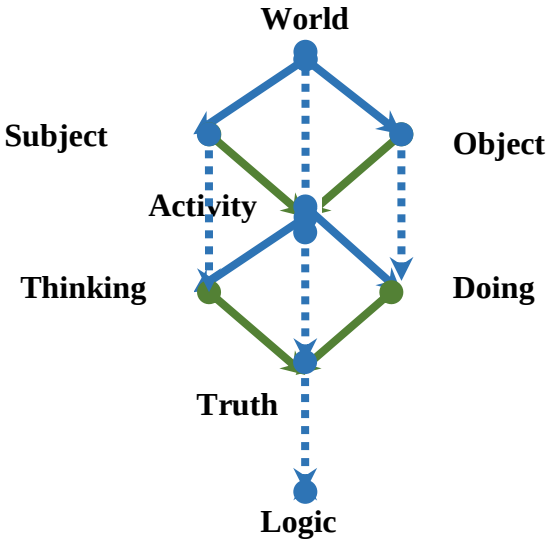

The next step is the specification of truth. This step is crucial, since it specifies the truth. Here we can use the conventional definition of truth as the correspondence of thinking to reality (to the object), and also the fact that the main thing for the subject is thinking as a sublated subject. But what will be the sublated object? Since the main thing in the object is those actions that characterize it, then we will accept the thesis that this sublated object will be *doing*.

These considerations can be presented by the following double dialectical tetrad:



Doing is similar to Action but we would like to emphasize that Doing is opposite to Thinking while Action is related to Activity that has both aspects Thinking and Doing.

Finally, we add to the above diagram the concept Logic as the sublated Truth:




This gives the first definition of logic:

***Logic is the science of Truth***

The presented diagram is called *the Diagram of the basic concepts of logic.*

Summarizing, we can formulate the following interrelations between concepts of logic:

- Logic is the science of Truth (is sublated Truth)
- Truth is sublated Activity that reveals itself as a unity of Thinking and Doing

- Activity is sublated World that reveals itself as a unity of Subject and Object
- Thinking is sublated Subject and Doing is sublated Object

The concept of Activity is central, therefore the proposed approach can be called *activity-based*. Thus, for further development of concepts of logic, the thorough analysis of Activity is required, in particular levels of activity should be identified.

### *1.6. Levels of Activity*

The principle of development (assent) from the abstract to the concrete should also be applied to the concept of activity, in other words, it is necessary to identify levels of activity leading from simple to more complex.

We start with simple examples.

**Example 1.** Let us assume that a student received a message that his friend was arriving by train and the student decided to meet him at the station. The student's activities are as follows:

1) the student decided to go to the station by a taxi;
2) the student looks for a suitable taxi and makes an order;
3) the taxi arrives at the place of order;
4) the student gets into the taxi, chooses a suitable seat, asks to turn on the music etc.;
5) the taxi arrives at the station.

**Example 2.** Let us assume that a volunteer asked a refugee about her needs. Then the situation develops as follows:

1) the refugee asked the volunteer to provide her with a phone;
2) the volunteer analyzed the refugee's needs, the availability of money in his fund, the work of providers in that area and, and based on the results of this analysis, ordered a phone;
3) having received the parcel with the phone, the volunteer checks whether it is the phone he ordered;

4) then, together with the woman, they set up the phone;
5) the refugee calls her relatives and tells them the news.

Note that in these examples, five levels of activity are distinguished. This can be seen in other examples as well. From this, we can conclude that such levels are not distinguished by chance, but that there is a fundamental reason for their identification. Indeed, such levels can be distinguished from the analysis of Hegel's works on *being*. We apply these levels to the concept of activity and obtain the following levels of activity:

- pure;
- becoming;
- determinate;
- real;
- actual.

Let us briefly characterize these levels:

- pure level: there is no activity;
- becoming: activity consists of transitions from one to another, such transitions end with a certain (determined) object;
- determinate level: activity is associated with the quality, quantity, and measure of objects;
- real level: activity is associated with the properties and relationships of objects (things);
- actual level: activity is associated with the actions of things.

Now let us describe these levels in more detail.

**Pure level of activity.** Nothing happens, A remains the same A. Note that Aristotle's law of identity (A is A) is related to this level.

**Becoming level of activity.** The main activity is transitions, first from pure being **B** to nothing **N**, then transitions follow the scheme of the dialectical triad, when a synthesis of pure being **B** and nothing **N** leads to new object denoted **BN**, then a new synthesis appears that leads to object **NBN** etc. At the moment when transitions cease,

an object of the determinate level arises. We call such an object an item. This level includes two partial cases: identical transitions and involutive transitions, when a double transition returns to the initial object. In a more general case, we obtain finite and infinite automata (transition systems).

**Determinate level of activity.** At this level the main activity is choosing objects (items) from a certain class determined in the world. For such activity the main related categories are *quality*, quantity, and measure. According to the principle of the ascent from the abstract to the concrete, we start with the most abstract category of this level: with the category of quality. Hegel writes that "something has a quality, and in this quality it is not only determined but delimited; its quality is its limit and, affected by it, something remains affirmative, quiescent existence" [Hegel1969]. So, we adopt that at the level of determinate being objects can be characterized by their qualities. But note that if a certain quality of an object is changed then a new object appears. We stop here not going in further characteristics of the category of quality because the described characteristics are sufficient to move further. Summarizing, it is possible to say that we accept *the ontological assumption* that at the determinate level of activity objects are specified by their qualities which are used for choosing objects by their qualities.

**Real level of activity.** At the level of real activity, the main philosophical categories are *thing*, *property*, *relation* [Hegel1969]. The predecessor of the category of thing is the category of item from the determinate level. Here it is important that things (objects) can change, remaining themselves (self-identity), that is, we can talk about states of things. This is how a thing differs from an item. Considering that for a thing the main category is the category of property, the main type of activity is the choosing of states of a things or choosing a thing from a class of things according to their properties. Summarizing, it is possible to say that we accept the ontological assumption that at the real level of activity objects are specified by

their properties which are used for choosing objects or their states by their properties.

**Actual level of activity.** At the level of actual activity, the main philosophical categories are *necessity and possibility*, *action and reaction* [Hegel1969]. This level is a synthesis of the level of becoming activity with the levels of determinate and real activity. Thus, it is possible to say that we accept the ontological assumption that at the actual level of activity objects are specified by their actions, qualities and properties which are used for choosing objects or their states by their qualities, properties, and actions.

Thus, the levels of activity from simple to more complex have been described, now we will move on to another classification according to the level of generality of concepts: the most general are philosophical categories, then scientific concepts, and finally, mathematical concepts.

### *1.7. Three-level scheme of logical concepts*

The conducted consideration of the basic principles and schemes of logic clearly indicates the need of the philosophical categories associated with logic. On the other hand, the application of logic in science and in practice indicates the need to study scientific and mathematical concepts. Thus, the idea of considering three levels (philosophical, scientific and mathematical) in their interrelations for the disclosure of the basic concepts of logic, including compositional logics, arises.

Such a three-level approach permits to align terminology for various levels of logic, demonstrate interrelations between philosophical categories, scientific and mathematical notions, and construct logical systems for various application domains.

Short descriptions of the levels are as follows.

**Philosophical level**

This level aims to present the most general features of objects. Such features are described by a system of *categories*. This problem belongs to such branch of philosophy as gnoseology (epistemology). The first system of categories is attributed to Aristotle. Later, Kant extended his system. Here we advocate the usage of a system developed by Hegel [Hegel1969]. He treats categories as *infinite* notions. Categories are grouped by stages of being:

- *pure* being;
- *becoming* being;
- *determinate* being;
- *real* being;
- *actual* being.

There are many categories related to these stages but we identify only those categories that evolve to the notions of ML related to AI:

- pure being: pure being, nothing;
- becoming being: becoming,
- determinate being: *quality*, *quantity*, and *measure*;
- real being: *thing*, *property*, and *relation*; *form* and *content*; *whole* and *part*;
- actual being: *action* and *reaction*; *cause* and *effect*; *ends* and *means* (*goals* and *tools*); *necessity* and *possibility*.

We treat these levels of being as levels of activity.

Let us note that the importance of philosophical foundations for logic is widely recognized [PhilLog2006].

**Scientific level**

At the second (scientific) level the basic scientific notions for applied domains are introduced. To do this we choose the categories related to logic and "finitize" them to the scientific concepts (notions). We start with the categories *universal*, *particular*, and *indi-*

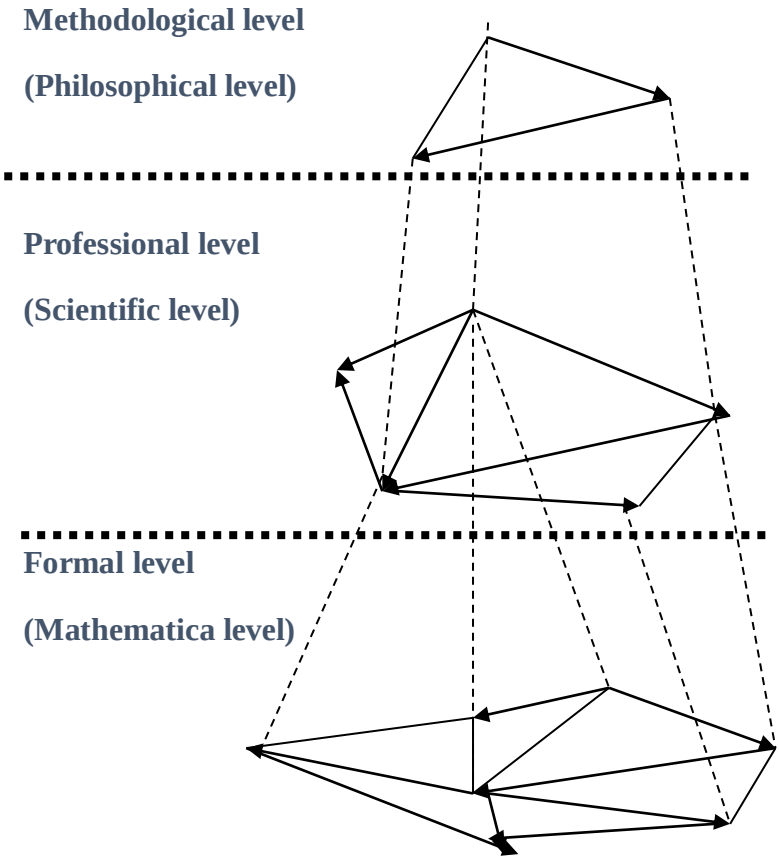
*vidual*. Their projections to the scientific level give the notions of *intension* and *extension*. This terminology stems from the set-theoretic treatment of extensionality that defines two sets equal if they consist of the same elements. We adopt and follow *the principle of intension–extension integrity*: a notion should be presented by the following diagram: *notion intension – notion especialty – integrity of intension, especialty, and extension; the intension in this integrity plays a leading role*. This principle may be considered as a generalization of Frege's meaning triangle.

**Mathematical level**

At the mathematical (formal) level of consideration, the notions from the previous level are restricted in order to get their reasonable formalization. This formalization should take into account intensional and extensional aspects of notions. Traditionally mathematical formalization is based on the notion of set. This notion is defined in extensional manner. But now we can see more examples that indicate on necessity of adding intensional aspects to the notion of set [*Nikitchenko, Chentsov*, 2012]. Still, even being extended, the notion of set is not expressive enough to represent numerous intensional aspects. Therefore, we propose to base mathematical formalization not only on the notion of set but also on the notion of function (mapping). Of course, this notion of function should be extended with intensional aspects. Such considerations give the

*principle of intensionalized function-theoretic formalization of the notions of mathematical logic*.

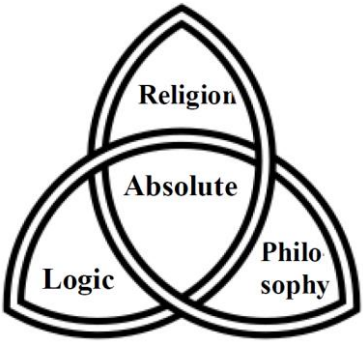
Thus, we obtain the following "pyramid" (network) of categories and concepts:




Transition from philosophical level to scientific level is called *finitization* and from scientific to mathematical level is called *formalization.*

Let us make some remarks regarding the philosophical level. At this level, categories are considered infinite. This means that it is impossible to precisely separate one category from another, because

if it were possible to do so, the categories would turn into finite concepts. Also note that categories, in addition to philosophy and logic, are also considered infinite (absolute) in religion. This explains the closeness of the three areas: philosophy, logic, and religion, as presented in the following figure:




### *1.8. Methodological basis: summary*

The general characterization of the three-level activity-based approach can be summarized as follows:

- Integration of logics
- By Development
- From the Abstract to the Concrete
- From
  - Methodological (philosophical) via
  - Professional (scientific) to
  - Formal (mathematical) level
    (vertical integrity)
- With Internal Integrity on each Level
  (horizontal integrity)

In the chapter the methodological basis of compositional logics has been developed, in particular:

- Conventional definitions of mathematical logic were considered and criticized;
- The main methodological principles and schemes were formulated;
- The main concepts of logic and their interrelations were presented; logic is defined as the science of truth;
- Levels of activity were distinguished.

### *1.9. Questions and exercises*

1. What are conventional definitions of logic?
2. What are conventional definitions of mathematical logic?
3. What are the difficulties in conventional definitions of logic?
4. What are the dictionary definitions of the word ‘logic’?
5. What are the synonyms of the word ‘logic’?
6. What are the antonyms of the word ‘logic’?
7. What are the initial characteristics of logic?
8. What is ‘infinite’?
9. What is the relationship between the infinite, the universal and the absolute?
10. What is the relationship between the finite and the infinite?
11. Write an essay on Zeno's aporias. How are they related to logic?
12. What is the preliminary definition of logic?
13. What does the naive approach mean?
14. What is the most important thing of a man?
15. Write an essay on the category of activity.
16. What are the definitions of truth?
17. Write an essay on the types of truth.
18. What are the main characteristics of science?
19. Write an essay on the types of values (goodness).
20. What are the relationships between science and values?
21. What are the relationships between logic and values?
22. What is generalization and specialization?

23. What is the hierarchy of the basic concepts of logic?
24. What is methodology?
25. Formulate the principle of universal interrelation, its consequences and application.
26. Formulate the principle of the ascent from the abstract to the concrete, its consequences and applications.
27. What is the abstract and the concrete, their interrelation?
28. What is sublation?
29. What is a dialectical triad?
30. What is a dialectical tetrad?
31. What is the diagram of the main logics and their interrelations?
32. What are the main methodological schemes?
33. Write an essay on "The science of logic" by W.G.F. Hegel.
34. What is a subject and an object?
35. What is thinking and doing?
36. What is the subject of logic?
37. Describe the levels of the activity category.
38. Describe the three-level scheme for explication of the concepts of logic. What are the interrelations of the levels?
39. What are the general characteristics of the three-level approach?
40. What are the main philosophical categories associated with logic?

## Chapter 2. Constructing a hierarchy of compositional logics

We construct a hierarchy at scientific and mathematical levels starting with abstract logic.

### *2.1. Abstract logic*

Let us move on to the scientific and mathematical construction of abstract models. We will use mathematical concepts and notations.

According to the Diagram of the basic concepts of logic, Logic is the science of Truth, and Truth is the unity of Thinking and Doing. Therefore, first we need to build a mathematical model of Truth, which will be the unity of models of Thinking and Doing.

When constructing such models, we take into account the following remarks:

– all categories are defined as a unity of the universal, the particular and the individual, and, since we are constructing the simplest models, then at the beginning the main emphasis is placed on the individuals in the form of classes or sets;
– usually we will distinguish between the terms ‘class’ and ‘set’. Such a distinction has a thousand-year history. Let us limit ourselves to a simple remark: a class is something very large, which is difficult to describe. A set, on the contrary, is clearly defined. For example, a class of all worlds, a class of all objects. It is difficult to say what it is, what is the cardinality of this classes. A set of formulas is clearly defined, its cardinality is definite, it depends on the cardinality of the set of symbols (alphabet).

How to build the simplest model of doing? Since these are sublated objects, we need to consider the simplest models of doing. And this is pure doing, i.e. there are no deeds (actions), thus the simplest model of doing is a class of models of objects with pure deeds. In this case, this class is identified with the models themselves, just as

zero-ary functions (constants) are identified with the constants themselves.

Therefore, we take the most abstract interpretation, that we have a certain class of abstract models of objects, that is, simply a certain class $M$ of models (models of worlds, worlds), about which nothing is known.

Let us now focus on building an abstract model of thinking. Again, we do not yet know what thinking is, but (based on practice) we see that its simplest form is a proposition (judgment, statement). We will model such propositions as formulas. Therefore, the simplest abstract model of thinking will be a certain set $Fr$ of formulas.

It remains to build a model of truth. Since truth is the unity of thinking and doing, the simple model of truth (denoted by $\models$) will be a certain binary relation on the class of models $M$ and the set of formulas $Fr$, i.e. $\models \subseteq \boldsymbol{M} \times \boldsymbol{Fr}$. We see that truth (the truth relation) really combines the models of doing and thinking. The truth of the formula $\Phi \in \boldsymbol{Fr}$ in the model $w \in \boldsymbol{M}$ is denoted by $w \models \Phi$ or ${}_w \models \Phi$ (infix form).

In this way, three models are constructed: a model of doing, a model of thinking, and a model of truth. The resulting tuple defines an *abstract truth system* (*ATS* ) or logic $\tau = (M, Fr, \models)$.

All the information in *ATS* actually is just some abstract binary relation $\models$. Can there be any theory on such poor material?

At first glance, it seems that nothing important can be built. However, this is not so. Even on such poor material, relying on practice, one can build quite meaningful logic systems.

Let us move on to the development of definitions and properties of *ATS*, relying on practice and the method of the ascent from the abstract to the concrete.

We discard empty models from consideration, that is, models in which $M=\varnothing$ or $Fr =\varnothing$ or $\models = \varnothing$. So, we will consider a system $\tau = (M, Fr, \models)$, for which $M \neq \varnothing$ and $Fr \neq \varnothing$ and $\models \neq \varnothing$.

Let us define the coverage of the set of formulas $Fr$ by the following sets of formulas:

– the set *Taut*(τ) of formulas that are true everywhere in *M* (identically true formulas, tautologies),

– the set *Cont*(τ) of formulas that are false everywhere in *M* (contradictions),

– the set *Sat*(τ) of formulas that are true in *M*.

Namely:

– the formula $\Phi \in Fr$ is called *always true* in τ if it is true for all models $w \in M$, i.e. $w \models \Phi$ for all models $w \in M$, denoted $M \models \Phi$ or simply $\models \Phi$ if *M* is known from the context.

– the formula $\Phi \in Fr$ is called universally false (*contradiction*) in τ if it is not true for all models $w \in M$, i.e. $w \not\models \Phi$ for all models $w \in M$, denoted $M \not\models \Phi$ or simply $\not\models \Phi$ if *M* is known from the context.

– the formula $\Phi \in Fr$ is called *satisfiable* in τ if it is true for some model $w \in M$, i.e. $w \models \Phi$ for some models $w \in M$, denoted $M \sim \Phi$ or simply $\sim \Phi$ if *M* is known from the context.

The above definitions allow us to write that

– $Taut(\tau)=\{\Phi \in Fr \mid \forall\, w \in M\ (w \models \Phi)\}$,

– $Cont(\tau)=\{\Phi \in Fr \mid \forall\, w \in M\ (w \not\models \Phi)\}$,

– $Sat\ (\tau)=\{\Phi \in Fr \mid \exists\, w \in M\ (w \models \Phi)\}$.

Let us give some additional definitions based on the concept of the image of the formula Φ with respect to the relation $\models$. We will denote such an image by Mod(Φ) and call it the class of models Φ. Thus,

$$Mod(\Phi)=\{w \in M \mid w \models \Phi\}.$$

In the introduced terms

– $Taut(\tau) = \{\Phi \in Fr \mid Mod(\Phi)=M\}$,

– $Cont(\tau) = \{\Phi \in Fr \mid Mod(\Phi)=\varnothing\}$,

– $Sat\ (\tau)\ = \{\Phi \in Fr \mid Mod(\Phi)\neq\varnothing\}$.

**Lemma 1.1.** (properties of Taut(τ), Cont(τ), Sat(τ)). For arbitrary logic τ (abstract truth system), the following properties hold:

– $\mathrm{Taut}(\tau) \cap \mathrm{Cont}(\tau) = \varnothing$,

– $\mathrm{Sat}(\tau) \cap \mathrm{Cont}(\tau) = \varnothing$,

– $\mathrm{Taut}(\tau) \subseteq \mathrm{Sat}(\tau)$,

– $\mathrm{Sat}(\tau) \cup \mathrm{Cont}(\tau) = Fr$.

Proof follows from the definitions.

The proposed definitions and theorems belong to *logic* because logic is the science of truth. This logic is denoted $L^{\tau}$.

The truth relation can be generalized to the relation of conditional truth (consequence relation) by introducing premises (conditions, hypotheses) from which conclusions follow. We call such a generalization the relation of *consequence* |= in models. This is a semantic concept, because it appeals to extralinguistic entities (to models of objects).

The main types of consequence are Hilbert's $\models_{\mathrm{H}} \subseteq \boldsymbol{M} \times B(Fr) \times Fr$ (the consequence is one formula) and Gentzen's $\models_{\mathrm{G}} \subseteq \boldsymbol{M} \times B(Fr) \times B(Fr)$ (the consequence is a set of formulas). In the infix form in the first case we write $\Gamma \, {}_{w}\!\models_{\mathrm{H}} \Phi$, where $\Gamma \subseteq Fr$ is the set of premises, $\Gamma \in Fr$ is the conclusion, $w \in \boldsymbol{M}$, in the second case – $\Gamma \, {}_{w}\!\models_{\mathrm{G}} \Delta$, where $\Delta \subseteq Fr$ is the set of conclusions.

Since logic studies the universal (the absolute), the question arises about the subclass of formulas that specifies the universal. This leads to the concept of logical truth (tautologies) and logical consequence. Namely, a consequence is logical in τ if it is fulfilled for all models. Then we write $\Gamma \models_{\mathrm{H}} \Phi$ ($\Gamma \models_{\mathrm{G}} \Delta$) if $\Gamma \, {}_{w}\!\models_{\mathrm{H}} \Phi$ ($\Gamma \, {}_{w}\!\models_{\mathrm{G}} \Delta$) for all $w \in \boldsymbol{M}$. If the type of consequence is clear from the context, then the indices H and G are not written. The concept of consequence (generalized truth) is a semantic concept. Logical conse-

quences serve as the basis for formulating the rules of inference (deduction, conclusion).

Next, we will mainly consider the Gentzen-type relation of consequence. (Investigate the corresponding properties.)

It should be noted that a logical consequence is a relation on sets of formulas, i.e. $\models_G \subseteq B(Fr) \times B(Fr)$, in which the class of models is not explicitly present. This class is implicitly present, because the definition of a consequence is based on a consequence in models. Therefore, the difficulty lies in the fact that the definition of a logical consequence (which is the law of thinking and doing) is difficult to verify. This leads to the idea that it is necessary to formulate simpler methods for defining a logical consequence that do not rely on the class of models. The resulting new relation is called *inference*.

Inference relation $\vdash$ is a syntactic concept of $L^{\tau}$ logic, which aims at syntactic formalization of the semantic concept of a logical consequence $\models$. Indeed, if truth is interpreted as related to doing, and derivability as related to thinking, then it is necessary to achieve the unity of thinking and doing for logic to be "true". The concept of derivability is specified on the basis of the concept of logical calculus (deductive system, formal system, axiomatic system).

There are two main types of inference (derivability) relation: Hilbert's $\vdash_H$ and Gentzen's $\vdash_G$, which are specified on the basis of the corresponding logical calculus (will be given below). We also note that in addition to the above, there are other types of consequence relations (natural, resolution, intuitionistic, etc.).

By a deductive system (axiomatic system, logical system, formal system, etc.) we mean the tuple $\delta = (Fr, Ax, R, [PR])$, where *Fr* is the language of the formal system, *Ax* is the set of axioms, R is the set of derivation rules, *PR* is the restriction on proofs, which specifies the derivability relation $\vdash_\delta$. If the set of restrictions is clear from the context, then the system is represented as a triple (*Fr*, *Ax*, *R*). The purpose of an axiomatic system is to formalize proof methods.

Now it is possible to define abstract logic in a deductive-theoretic style (proof-theoretic style, style of proof theory).

Namely, abstract deductive logic $L^{\delta}$ is a tuple ( *Fr*, $\vdash_{\delta}$), where $\delta$ = (*Fr*, *Ax*, *R*, [*PR*]) is a certain deductive system (we do not write the index DS if it is clear from the context). Deductive systems are studied in the theory of proofs.

So, the logic $L^{\tau}$ is built on the basis of the theoretical-model approach (model theory), and $L^{\delta}$ is based on the theoretical-deductive approach (proof theory). Thus, we have two logics that are associated with doing and thinking, respectively. This prompts us to consider the integrated abstract logic $L^{\tau,\delta}$ = (*M*, *Fr*, $\models$, $\vdash$).

Abstract logic:

– is an initial logic;

– it introduces the basic concepts into consideration;

– it is abstract because the class of world models and language are given abstractly (i.e. there are no structures of elements of these classes), the other two concepts – truth (consequence relation) and derivability are given schematically (i.e. only the type of these concepts is indicated);

– despite its abstractness, it allows to classify logics according to the levels of (generalized) truth and derivability, as well as to introduce derived concepts and to give certain equivalent forms to already introduced concepts.

The introduction of the integrated abstract logic allows to formulate the main problem of mathematical logic: this is the clarification of its basic concepts and the study of their relationship at different levels of abstractness and concreteness.

Let us first consider the relations of truth and derivability (i.e. the relationship of semantics and syntax of logic). Logical truths (logical consequences) are called *laws* of logic, and the proved formulas are called *theorems*. The class of logical truths (laws) is denoted by *Truth*, and the class of theorems is denoted by *Theorem*. Logic *L* is called *correct* (valid) if every theorem is a truth (law), i.e. *Theorem* $\subseteq$*Truth*, logic is called *complete* if every truth (every law) is a theorem, i.e. *Truth* $\subseteq$*Theorem*, logic is called *adequate* if the sets of the-

orems and truths (laws) coincide, i.e. *Truth =Theorem*. Note that there should be a specific unity of truths (laws) and theorems in logic. Such unity (of doing and thinking) is very important, without it logic will be "untrue", with limited possibilities.

The most important derived concept is the concept of the interpretation of formulas. This concept naturally arises from the truth relation. Indeed, such a relation has the following type: $|= \subseteq \boldsymbol{M} \times Fr$. If we fix some formula in this relation, say $\Phi$, then for an arbitrary model $w \in \boldsymbol{M}$ we get two possibilities: ${}_w|=\Phi$ or ${}_w|\neq\Phi$ (i.e. ${}_w\Phi$ is true on $w$ or not true). This means that the truth on a fixed formula determines a mapping that assigns the values 1 or 0 (T or F) to specific models, i.e. takes values on the set {1, 0} (the set *Bool*). Such a mapping is called a predicate, and the mapping of each formula to the corresponding predicate is called the *interpretation I* of formulas in the logic *L*, i.e. $I$: $Fr \to Bool^M$. The concepts of interpretation and predicate are semantic concepts. In particular, a predicate expresses qualities or properties of models. Thus, the concepts of truth of a formula and its interpretation are equivalent (at this level of abstraction).

In the general case, a predicate on a set *A* is understood as an arbitrary mapping $P$: $A \to \{1, 0\}$; the set of predicates is denoted by $\{1, 0\}^A$ ($=Bool^A$). Operations on predicates are specified in *predicate algebras*. Note that in linguistics, philosophy and some other sciences the term predicate can be interpreted in a different way, say as a characteristic of a subject.

Conclusion: abstract logic, despite its abstractness, allows us to define the basic concepts of mathematical logic and formulate its main problems. The further development of mathematical logic is associated with the consideration of specific logics induced by the levels of activity.

### *2.2. Propositional logic as a concretization of abstract logic*

Propositional logic (logic of propositions) investigates the problem of choosing a class of objects at the level of determinate activity according to their qualities. This logic can be defined as independent of other logics. But we will define it as a concretization of abstract logic. This will allow us to establish the connections of propositional logic with other logics and more clearly indicate its role in the system of mathematical logics.

We recall that the abstract mathematical logic is considered as a tuple ($M$, $Fr$, $\models$, $\vdash$), where $M$ is a class of models (of objects, worlds), $Fr$ is the language of logic (set of formulas), $\models$ is a relation of consequence (generalized truth), $\vdash$ is a relation of derivability (inference, provability). Let us construct propositional logic $\boldsymbol{L}^{Pr}$ as a concretization of abstract logic.

Class $M$ of models of worlds of propositional logic. At the level of determinate being and activity, only the qualities of objects are known. Therefore, the model of an object (propositional world) is a set of values (characteristics) of qualities. We will specify qualities by propositional variables (symbols, names) from the set $Ps$. Therefore, an object (world) is characterized by a set of variables with their values. There are different ways of representing such sets: tabular, nominative, relational, functional, etc. For example, let $Ps=\{a, b, c\}$, take an object (world) with existing qualities $a$ and $c$, but with the absence of $b$. Then for such an object we have:

– Tabular representation (as a table row with qualities as column headings):

| $a$ | $b$ | $c$ |
|---|---|---|
| 1 | 0 | 1 |

– Nominative representation (as a set of named values from the set $\{1, 0\}^{Ps}$ $(= Bool^{Ps})$):

$$[a \mapsto 1, b \mapsto 0, c \mapsto 1]$$

– Relational representation (as a binary relation on $Ps \times \{1, 0\}$ $(= Ps \times Bool)$):

$$\{(a, 1), (b, 0), (c, 1)\}$$

– Functional representation (as a function from *Ps* into {1, 0}, i.e. as an element of the set $Ps \to \{1, 0\}$ $(= Ps \to Bool = Bool^{Ps})$. Such a function can be represented by the following tests: on arguments *a* and *c* it gives the value 1, on argument *b* it gives the value 0. Its graph is a relational representation (see above).

Thus, at this level $M = Bool^{Ps}$. The elements of *M* are called Boolean *Ps*-sets or simply Boolean set.

**The language of propositional logic.** This language arises naturally from the problem of describing classes of propositional worlds:

– we need to indicate that a certain object simultaneously has certain qualities – this is the operation of conjunction,

– then we need to indicate the absence of a certain quality – this is the negation of its presence,

– it remains to indicate those objects that are included in the required class – this is disjunction.

Thus, we have connections (operations) in the language: conjunction $\wedge$, negation $\neg$, disjunction $\vee$. Note that you can remove conjunction or disjunction; this will not affect the expressive power of the language. From here we get an inductive definition of the language (as a set of *FPr* propositional formulas):

– if $P \in Ps$, then $P \in FPr$; such formulas are called *atomic*;

– if $\Phi, \Psi \in FPr$, then $\Phi \vee \Psi, \Phi \wedge \Psi, \neg\Phi \in FPr$; such formulas are called *complex* (*structured*).

Thus, $Fr = FPr$. Elements of *FPr* are called propositional formulas.

Note: other operations can be added to the language – implication, equivalence, etc.

**Truth in propositional logic**

Initially (in abstract logic), truth was defined as the relation $\models \subseteq \boldsymbol{M} \times Fr$. This relation can be represented by the interpretation mapping *I*: $Fr \to Bool^{\boldsymbol{M}}$. At the propositional level $M = Bool^{Ps}$, $Fr = FPr$

therefore, the interpretation mapping at the propositional level $I^{Pr}$ will be the mapping $I^{Pr} : FPr \to Bool^{Bool^{Ps}}$. Let us analyze the specified formula: in it $Bool^{Bool^{Ps}}$ is the mapping $Bool^{Ps} \to Bool$, which associates each propositional world (model) with a certain truth value – and therefore, it is a predicate (Boolean function). So, the mapping of the interpretation $I^{Pr}$ associates each propositional formula with a certain predicate – a certain Boolean function.

The presence of structured propositional formulas now provides the possibility of inductively defining the mapping $I^{Pr}$. According to the principle of structural induction, it is first necessary to define the interpretation of the atomic formula $P$ ($P \in Ps$). Such a formula indicates the presence of the quality $P$ in a given world $w$. Since worlds are Boolean sets, which we interpret them as functions $Ps \to Bool$, (i.e. $w \in Ps \to Bool$), then $I^{Pr}(P) = w(P)$. Next, we apply induction for complex formulas:

– for the formula $\Phi \vee \Psi$ we have $I^{Pr}(\Phi \vee \Psi) = I^{Pr}(\Phi) \vee I^{Pr}(\Psi)$;

– for the formula $\Phi \wedge \Psi$ we have $I^{Pr}(\Phi \wedge \Psi) = I^{Pr}(\Phi) \wedge I^{Pr}(\Psi)$;

– for the formula $\neg\Phi$ we have $I^{Pr}(\neg\Phi) = \neg I^{Pr}(\Phi)$.

There is a certain "trick" in the above formulas, which must be clearly understood. Namely, the symbols of propositional relations ($\wedge, \neg, \vee$) have different meanings in the left and right parts of the above formulas: in the left part they are simply syntactic signs, and in the right part – symbols of operations on predicates (we call such operations *compositions*). Accordingly, we obtain an algebra of predicates with compositions $\wedge, \neg, \vee$. We will not formally describe such algebras here. It should also be said that the indicated symbols are also traditionally understood as symbols of the corresponding operations on Boolean values (Boolean numbers); these operations are given by the truth tables.

The introduced notations allow, finally, to determine the truth of the formula $\Phi$ in the world $w$: $_w \models \Phi$ if and only if the value $I^{Pr}(\Phi)(w)$ is true. Accordingly, $\Phi$ is a formula that is true every-

where if it is true for all *w*. In propositional logic, formulas that are true everywhere are called tautologies. These are the laws of propositional logic.

Let us give examples of tautologies ( $\Phi, \Psi$ – propositional formulas).

De Morgan's laws:

- $\models \neg(\Phi \wedge \Psi) \sim \neg\Phi \vee \neg\Psi$ ;
- $\models \neg(\Phi \vee \Psi) \sim \neg\Phi \wedge \neg\Psi$ .

Modus ponens rule: $\Phi, \Phi \to \Psi \models \Psi$ .

**Derivability in propositional logic.**

We recall that a deductive system (logical calculus, formal system, axiomatic system, etc.) is understood as a tuple (*Fr*, *Ax*, *R*, [*PR*]), where *Fr* is the language of the formal system, *Ax* is the set of axioms, *R* is the set of inference rules, *PR* is the restrictions of proofs.

Let us now consider the sequent type calculus for propositional logic. The set of formulas is *FPr*.

The set of axioms *Ax* is given by closed sequences, i.e. such marked sequences that there exists a formula $\Phi$ such that ${}_1\Phi$ and ${}_0\Phi$ belong to the sequent (for unmarked sequents, this is a closure condition that means the presence of a formula $\Phi$ that appears in the left and right parts of the sequent).

The set of rules *R* consists of the following sequent rules (for operations $\vee, \neg, \to, \leftrightarrow$ ):

$${}_1\neg\ \frac{{}_0A, \Sigma}{{}_1\neg A, \Sigma} \qquad {}_0\neg\ \frac{{}_1A, \Sigma}{{}_0\neg A, \Sigma}$$

$${}_1\vee\ \frac{{}_1A, \Sigma \quad {}_1B, \Sigma}{{}_1A \vee B, \Sigma} \qquad {}_0\vee\ \frac{{}_0A, {}_0B, \Sigma}{{}_0A \vee B, \Sigma}$$

$${}_1\wedge\ \frac{{}_1A, {}_1B, \Sigma}{{}_1A \wedge B, \Sigma} \qquad {}_0\wedge\ \frac{{}_0A, \Sigma \quad {}_0B, \Sigma}{{}_0A \wedge B, \Sigma}$$

$$ {}_1\!\to \frac{{}_0A,\Sigma \qquad {}_1B,\Sigma}{{}_1A\to B,\Sigma} \qquad {}_0\!\to \frac{{}_1A,\ {}_0B,\Sigma}{{}_0A\to B,\Sigma} $$

$$ {}_1\!\leftrightarrow \frac{{}_0A,\ {}_0B,\Sigma \qquad {}_1A,\ {}_1B,\Sigma}{{}_1A\leftrightarrow B,\Sigma} \qquad {}_0\!\leftrightarrow \frac{{}_0A,\ {}_1B,\Sigma \qquad {}_1A,\ {}_0B,\Sigma}{{}_0A\leftrightarrow B,\Sigma} $$

In the sequent calculus proofs are trees (see in the chapter 6).

**Tautology theorem for propositional logic**: the set of theorems of propositional logic coincides with the set of tautologies. That is, every theorem is a tautology and vice versa, a tautology is a theorem.

The tautology theorem testifies the adequacy of logic, that is, the completeness and correctness of the logical means of propositional logic. This means that in logic, general (global) truth has been achieved as the unity of the laws of thinking (theorems with their proofs) and general goals of action (tautologies).

From the tautology theorem directly follows:

1) solvability of propositional logic: the set of theorems is algorithmically solvable with respect to the set of all propositional formulas;

2) consistency of propositional logic: there is no propositional formula $A$ such that $\vdash A$ and $\vdash \neg A$.

Indeed, if $\vdash A$ and $\vdash \neg A$, then $A$ and $\neg A$ are simultaneously tautologies.

Thus, for propositional logic, the basic problems of mathematical logic are solved positively.

### *2.3. Predicate logic as a concretization of propositional logic*

At the level of real being, the main categories for which certain formalizations will be built are the categories Thing and Property. Therefore, the world (which was considered abstractly in abstract logic) will now be a thing (the term object can also be used). Things have their own properties, which are characterized by certain parameters. The names of the parameters belong to a certain set of subject names (subject variables) *V*. For simplicity, we assume that all parameters take the same set *A* of values (this does not significantly limit the logic). (Note that the symbol *A* is often used in other meanings, for example, *A* can denote a certain formula.) A thing can have different parameter values. We will call the set of (named) parameter values as the state of the thing. Such a set is also called a (quasi-) nominative set. Therefore, with each thing we associate the space (class) of sets of its parameters, that is, the class of nominative sets, each of which defines a certain state of the thing. Mathematically, this will be the class $V \to A$, which we also denote $A^V$.

With the introduction of states, the formalization of a certain thing is not yet complete. It is necessary to specify the properties of states. Here, the property is an analogue (is a development) of the concept of quality from the previous level of logic. Therefore, it is necessary to introduce a set of property symbols, for which we will take the same notation as for the names of qualities of the propositional level – *Ps*. Now the property can be defined as a certain predicate from the set $A^V \to Bool$ $(= (V \to A) \to Bool)$. Let us denote this class of predicates $Pr_A^V$. We call them as predicates with variables, or quasiary predicates. It remains for us to introduce operations (compositions) on predicates. Compositions $\vee, \neg, \to, \leftrightarrow$ are analogous to compositions of the propositional level. But at the propositional level, the sets were Boolean (i.e., the names had only two values), and now the values belong to an arbitrary set *A*. To take this into account, the parametric existential quantifier $\exists x$ and the universal quantifier $\forall x$ are introduced with the composition parameter $x \in V$. Logic with such compositions is called pure first-order predicate logic, because here we do not consider functions over *A*. We

also note that it is enough to consider only compositions $\vee, \neg, \exists x$ – other compositions are derivatble. As for the basic (initial) predicates, we take the usual n-ary predicates.

Thus, at the level of real being, a thing is modeled (taken as a simple thing) as an algebra $\Re_A^V = (Pr_A^V; \vee, \neg, \exists x)$ whose carrier is a set of predicates over sets of named parameters (over quasi-nominative sets), and the operations (compositions) are disjunction, negation, and the existence quantifier.

The formulated provisions allow us to proceed to the definitions of the logic at the real level $L^{FO}$ – the first-order logic of quasiary predicates.

**Class *M* of models (worlds) of predicate logic.**

The analysis of worlds at the level of real activity allows us to specify the class of worlds *M* as a class of algebras $\Re_A^V$ with different values of parameters *A*, but with a fixed set *V*, i.e. $M=\{ \Re_A^V \mid$ for arbitrary $A\}$.

How to specify such algebras? It is impossible to do this with finite tables, because *A* can be infinite. Therefore, various methods will be used to describe *A*, primarily inductive or recursive definitions. The same problem arises for specifying compositions that should act "identically" in different algebras. We will not discuss this issue here.

**Language of first-order predicate logic.**

We will describe the language of pure predicate logic, i.e. a language without functional symbols. We fix the set of subject variables (parameter names) *V*, the set of symbols of compositions and the set of symbols of basic n-ary predicates *Ps*. In logic, such sets form the signature of the language. The class of formulas of the $L^{FO}$ language will be denoted $Fr^{FO}$ and called the set of formulas of first-order predicate logic. This class is inductively defined as follows.

– if $P \in Ps$, $x_1,..,x_n \in V$, then $P(x_1,..,x_n) \in Fr^{FO}$; such formulas are called atomic;

– if $\Phi, \Psi \in Fr^{FO}$, $x \in V$, then $\Phi \vee \Psi, \neg\Phi, \exists x\Phi \in Fr^{FO}$; such formulas are called complex.

Note: other operations can be added to the language – implication, equivalence, universal quantifier, etc.

The occurrence of a name (variable) $x$ in a formula $\Phi$ is bound if it is in the domain of some quantifier, otherwise such an occurrence of $x$ in $\Phi$ is free. If there is a free occurrence of the name $x$ in a formula $\Phi$, then $x$ is a free name (free variable) of the formula $\Phi$.

The formula $\Phi$ with free names $x_1,.., x_n$ is denoted by $\Phi(x_1,.., x_n)$. The formula is closed if it has no free names.

**Truth in first-order predicate logic.**

We will define truth through the concept of interpretation. But now this concept will be much more complicated than at the previous level. Let us explain it with an example. Let the formula $\Phi$ contain the sign "+", which is the sign of a certain binary operation. At the first step of the interpretation, it is necessary to understand in which algebra this sign will be interpreted. It can be the algebra of integers, or the algebra of real numbers, or the algebra of matrices, etc. For example, we will choose the algebra of natural numbers. But even in this case there can be different interpretations of "+" as a certain binary operation, for example, as traditional addition or as addition modulo 7, etc. In the first case, the value +(4, 5)=9, and in the second +(4, 5)=2. After the algebra and interpretation of the sign in this algebra have been chosen, we can give an interpretation to atomic formulas of the form $P(x_1,..,x_n)$. These formulas are interpreted as predicates with variables (quasi-predicates). The difference between such predicates and n-ary predicates is that the former are defined on sets of named values, and the latter on tuples of values, for example, for an expression $x+y$ the argument of the corresponding function can be a set $[x \mapsto 4, y \mapsto 5, z \mapsto 7,...]$, and for the expression +(4, 5) – the pair (4, 5). Similar examples can be given for predicate symbols.

Let us denote for a predicate symbol $P \in Ps$ its interpretation in algebra $\mathfrak{R}_A^V$ as a predicate with variables $I_A^V(\Phi)$. Further inductively:

– $I_A^V(\Phi \vee \Psi) = I_A^V(\Phi) \vee I_A^V(\Psi)$;

– $I_A^V(\neg\Phi) = \neg I_A^V(\Phi)$;

– $I_A^V(\exists x\Phi) = \exists x(I_A^V(\Phi))$.

As in the case of propositional logic, the symbols of operations on the right-hand side are symbols of compositions. Compositions are defined in the traditional way.

For propositional logic we had two levels of truth:

1) truth in the world and

2) truth in all worlds (logical truth).

At the level of first-order logic, the world is a collection of states (it is a set of named sets), so now there will be three levels of truth:

– truth of $\Phi$ in algebra $\mathfrak{R}_A^V$ and state *st* means that $(I_A^V(\Phi))(st) = 1$; we denote this by $\mathfrak{R}_A^V, st \models \Phi$;

– truth of $\Phi$ in algebra $\mathfrak{R}_A^V$ means truth of $\Phi$ in all states $st \in A^V$; we denote this by $\mathfrak{R}_A^V \models \Phi$;

– universal (logical) truth of $\Phi$ means truth of $\Phi$ in all algebras for different *A*; we denote this by $\mathfrak{R}_A^V$.

Here we can say that the first truth is singular, the second is particular, the third is universal.

We generalize the above concepts of truth to the concept of consequence.

Let us give examples of universally true formulas ( $P, Q$ – formulas of first-order predicate logic):

- $\exists x\,(P(x) \wedge Q(x)) \to (\exists x\,P(x) \wedge \exists x\,Q(x))$;
- $(\forall x\,P(x) \to \exists x\,Q(x)) \sim \exists x\,(P(x) \to Q(x))$.

**Derivability in first-order predicate logic**

Axiomatic systems of first-order logic are called first-order calculi, or first-order theories. By first-order theory we mean the formal system $T=(Fr^{FO}, Ax, R)$, where $Fr^{FO}$ is the language (set of formulas) of the first-order, *Ax* is the set of axioms, which is divided into a

set of logical axioms and a set of proper axioms, $R$ is the set of inference rules. We do not specify the set of proofs, because it was defined earlier. Logical axioms are present in all first-order theories, proper axioms determine the specificity of a particular theory.

For example, let us give sequent calculus.

The set of logical axioms in sequential-type systems is given by closed sequents.

The rules of inference (sequent forms) for compositions $\vee, \wedge, \neg, \exists x$ are as follows (conclusions are written above the line, premises are written below the line):

$$ {}_1\vee \ \frac{{}_1A \vee B, \Sigma}{{}_1A, \Sigma \quad {}_1B, \Sigma} \qquad {}_0\vee \ \frac{{}_0A \vee B, \Sigma}{{}_0A, \ {}_0B, \Sigma} $$

$$ {}_1\wedge \ \frac{{}_1A \wedge B, \Sigma}{{}_1A, {}_1 B, \Sigma} \qquad {}_0\wedge \ \frac{{}_0A \wedge B, \Sigma}{{}_0A, \Sigma \quad {}_0B, \Sigma} $$

$$ {}_1\neg \ \frac{{}_1\neg A, \Sigma}{{}_0A, \Sigma} \qquad {}_0\neg \ \frac{{}_0\neg A, \Sigma}{{}_1A, \Sigma} $$

$$ {}_1\exists \ \frac{{}_1\exists xA(x), \Sigma}{{}_1A(y), \Sigma} $$ provided that the new variable $y$ is not included in $\Sigma \cup \{\exists xA(x)\}$

$$ \frac{{}_0\exists xA(x), \Sigma}{{}_0A(z_1), ..., \ {}_0A(z_m), \Sigma, \ {}_0\exists xA(x)} $$ where $\{z_1, \ldots, z_m\}$ are all free variables; if there are no free variables, then an arbitrary new variable $z_1$ is taken.

Logical calculi (axiomatic systems) of first-order logics are also called first-order calculi, or first-order theories. The main result is Gödel's theorem.

**Theorem** (Gödel's completeness theorem). Let $T$ be a first-order theory. Then the formula $\Phi$ is true in $T$ ($T \models \Phi$) if and only if $\Phi$ is a theorem ($T \vdash \Phi$).

Since the correctness of the logical means of first-order theories is easily proven, Gödel's theorem actually asserts the adequacy of logic.

### *2.4. Modal logic as a concretization of predicate logic*

At the level of real being, the first-order logic of predicates was formulated, which serves as the initial formalization of the categories Thing and Property. At this level, there are many other categories that also require initial formalization. However, this will be quite difficult and will not correspond to the purpose of our initial course in compositional logic. Therefore, let's move on to the next level - the level of real being. The first (and easiest to formalize) are the categories Necessity and Possibility. Logics that consider such categories are modal.

It would seem that new categories require new approaches and methods for their formalization. At the same time, a more detailed analysis of the concepts of abstract mathematical logic allows us to identify moments of necessity and possibility already at this abstract level. Indeed, let's take a closer look at the concepts of universal truth and satisfiability of formulas. Universal truth $|=\Phi$ of a formula $\Phi$ means that it is necessarily true in all worlds (models), and satisfiability $|\sim\Phi$ of a formula $\Phi$ means its possibility of being true in a certain world. In abstract logic these concepts belong to the metalanguage (to the language of describing logic), and not to the object language (i.e., the language of logic itself). Therefore, in order to make abstract logic modal, these concepts must be introduced into logic itself, that is, the language must be enriched with the operators of modality “necessarily” (denoted by $\Box$) and “possibly” (denoted by $\Diamond$).

It would seem that there is no difficulty in enriching the language of abstract logic with such operators. But the language of abstract logic is truly abstract, formulas have no structure. Therefore, we will

introduce modal operators at the next level – at the level of propositional logic. In this case, we obtain propositional modal logic, which is denoted by the symbol *K* in honor of the logician Saul Kripke.

The language of logic *K* (the set of formulas $FM$ ) is given by the following rules:

– if $P \in Ps$ , then $P \in FM$ ;

– if $\Phi, \Psi \in FM$ , then $\Phi \vee \Psi, \Phi \wedge \Psi, \neg\Phi, \Box\Phi, \Diamond\Phi \in FM$ .

Note that one modal operator is enough, the other will be derived according to the following relations:

$$\Diamond\Phi = \neg\Box\neg\Phi \text{ and } \Box\Phi = \neg\Diamond\neg\Phi$$

(recall the relation between truth everywhere and feasibility).

In modal logics, the truth relation can be given in various ways, the most important of which is based on the reachability relation between worlds. Here we will limit ourselves to the simplest case, assuming that що $\Box\Phi \Leftrightarrow \models\Phi$ and $\Diamond\Phi \Leftrightarrow \models\sim\Phi$.

Let us now present axiomatic systems of Hilbert type for *K*.

The set of axioms of the system consists of axioms of propositional logic and axioms given by the normalization scheme

AxNr) $\Box(\Phi\rightarrow\Psi)\rightarrow(\Box\Phi\rightarrow\Box\Psi)$.

The inference rules consist of the inference rules of propositional logic, to which is added the modalization rule:

RM) $\Phi \vdash \Box\Phi$.

Modal logics can be generalized in different directions. Here are some examples.

System *T*. The axiom set consists of the axioms of system *K* and the axioms given by the scheme

Ax$\Box$) $\Box\Phi\rightarrow\Phi$.

The rules of derivation are the same as for system *K*.

System *B* (Brauer). The axiom set consists of the axioms of system *T* and the axioms given by the scheme

AxB) $\Phi\rightarrow\Box\Diamond\Phi$.

The rules of derivation are the same as for system *K*.

System *S*4 (Lewis). The axiom set consists of the axioms of system *T* and the axioms given by the scheme

AxS4) $\Box\Phi\rightarrow\Box\Box\Phi$.

The rules of derivation are the same as for system *K*.

System *S*5 (Lewis). The axiom set consists of the axioms of system *T* and the axioms given by the scheme

AxS5) $\Diamond\Phi\rightarrow\Box\Diamond\Phi$.

The rules of derivation are the same as for the *K* system.

Therefore, the description of the propositional modal system *S*5 consists of the following axioms and rules.

Axioms:

AxPr) $\neg\Phi\vee\Phi$ (propositional axiom);

AxNr) $\Box(\Phi\rightarrow\Psi)\rightarrow(\Box\Phi\rightarrow\Box\Psi)$;

Ax$\Box$) $\Box\Phi\rightarrow\Phi$;

AxS5) $\Diamond\Phi\rightarrow\Box\Diamond\Phi$.

Derivation rules:

R1) $\Phi \vdash \Psi\vee\Phi$ – expansion;

R2) $\Phi\vee\Phi \vdash \Phi$ – reduction;

R3) $\Phi\vee(\Psi\vee\Xi) \vdash (\Phi\vee\Psi)\vee\Xi$ – associativity;

R4) $\Phi\vee\Psi, \neg\Phi\vee\Xi \vdash \Psi\vee\Xi$ – cut;

RM) $\Phi \vdash \Box\Phi$ – modalization.

*Tautology theorem* (correctness and completeness) for propositional modal logics: for each of the above axiomatic systems, the set of theorems coincides with the set of tautologies.

Important variants of modal logics are temporal logics, which use such temporal operators as $\Box_{\uparrow}$ (always will be), $\Box_{\downarrow}$ (always was), $\Diamond_{\uparrow}$ (once will be), $\Diamond_{\downarrow}$ (once was), etc.

Modal logics in various versions are used for specification and verification of programs, in particular, a special program logic – dynamic logic [Harel1984] – can be defined as a multi-modal logic

with structured modalities. For each program π there are two parameterized modalities: a necessity modality [π] and a possibility modality <π>. For deterministic programs a special Floyd–Hoare logic is introduced by Hoare triad:

{precondition} program {postcondition}.

Such logics are considered in the next chapters.

### *2.5. Questions and exercises*

1. What concepts is abstract logic based on?
2. How is the concept of thinking formalized in abstract logic?
3. How is the concept of doing formalized in abstract logic?
4. How is the concept of truth defined in abstract logic?
5. What are the main properties of abstract logic?
6. How is the consequence relation defined in abstract logic?
7. What is an abstract system of truth?
8. What are the types of consequence relations?
9. What is a deductive system?
10. What are correctness (validity), completeness, and adequacy in abstract logic?
11. What categories is propositional logic associated with?
12. What is the main activity at the propositional level?
13. What is the model of an object at the propositional level?
14. What algebras are used in propositional logic?
15. How is the language of propositional logic defined?
16. How is truth defined in propositional logic?
17. How is an inference relation defined in propositional logic?
18. What is a tautology theorem for propositional logic?
19. What is an object at the level of real activity?
20. What algebras are used to formalize objects at the real level?
21. What is the language of first-order predicate logic?

22. How is truth defined in predicate logic?
23. How is inference relation defined in predicate logic?
24. How is Gödel's completeness theorem formulated?
25. What categories are used at the level of actual activity?
26. What new compositions appear at the level of actual activity?
27. What is model logic?
28. What logics can be considered as further concretizations of modal logic?

## Chapter 3. Predicate algebras and logics

In this chapter we consider different classes of predicate algebras that forms the base for various compositional logics.

### *3.1. Algebras of quasiary predicates*

Classical predicate logic is based on n-ary functions and predicates later used in construction of formulas with the help of logical connectives and quantifiers. But it is more adequate to treat semantics of formulas as a predicate defined on nominative data. Such predicates are called *quasiary predicates*.

**Example 3.1.** Consider a Boolean expression x<y. Its semantics can be formalized as a partial quasiary predicate less : $V^Z \xrightarrow{p}$ Bool. This predicate is undefined on a flat nominative data [x↦5, u↦4] (we write less([x↦5, u↦4])↑), is defined on [x↦5, u↦4, y↦2] with value *F* (we write less([x↦5, u↦4, y↦2])↓= *F*). Note that if a value of less is defined on some data, then the predicate is defined with the same value on any extension of this data. This property is called equitonicity (a special case of monotonicity). Thus, less([x↦5, u↦4, y↦2, v↦4])↓=*F*, *x*, *u*, *y*, *v*∈*V*. A specific new composition, which can be defined on this level, is renomination $R^{v_1,\dots,v_n}_{x_1,\dots,x_n}$ (formal definition will be given later). For example, when we ask how to represent a formal model of y<v given a formal model less of x<y, the answer will be $R^{x,y}_{y,v}$(less). The constructed predicate can be evaluated in the expected manner, e.g. $R^{x,y}_{y,v}$(less)([x↦5, u↦4, y↦2, v↦4]) = less([x↦2, u↦4, y↦4, v↦4]) = *T*.

Note that renomination (primarily in syntactic aspects) is widely used in classical logic, lambda-calculus, and specification languages like Z-notation [Spivey1992], B [Abrial1996], TLA [Lamport2002], RAISE [RAISE1992] etc.

The notion of quasiary predicate can also be easily understood when we analyze Tarski's definition of first-order language semantics [Mendelson1966]. This semantics is based on the notion of interpretation which consists of two parts:

1) interpretation of predicate and function symbols in some structure (algebra), and

2) interpretation of individual variables in the domain of this structure.

The latter are usually called variable assignments (or valuations) and can be represented by *total* mappings from a set of individual variables (names) $V$ into some set of basic values $A$. The class of such total mappings will be denoted $V \xrightarrow{t} A$ or $A^V$, and called total nominative sets. Thus, Tarski's semantics interprets predicate and function symbols as total quasiary predicates and functions defined on $A^V$. In applications like model checking, program verification, automated theorem proving, etc., *partial* assignments (nominative sets) are often used instead of total assignments. This means that predicate and function symbols can be interpreted as partial functions defined on the class $A^V$ of nominative sets with values in Bool and $A$ respectively, i.e. as quasiary predicates and functions.

More elaborated programming languages work with hierarchical nominative data. Composite names like x1.x2. … .xn are used to access data components in such languages. Such data can represent complex data structures used in programming.

Having defined the notion of quasiary predicate we can now start corresponding developing semantics-based logics. Such logics will be called *composition-nominative logics* (CNL). Analysis shows that the main semantic notion of mathematical logic – the notion of predicate – can be defined at different levels. At the Boolean level predicates are considered as partial functions from a class of abstract data

*Dt* to *Bool*. In this case such compositions as disjunction $\vee$, negation $\neg$, etc., can be defined. These compositions are derived from Kleene's strong connectives [Kleene1952] when partiality of predicates is taken into consideration. Thus, the main semantic objects are classes of algebras of partial predicates of the form $<Dt \xrightarrow{p} Bool; \vee, \neg>$. The obtained logics may be called *propositional logics of partial predicates*. Such logics are rather abstract, therefore their further development is required at the nominative level. At this level we have two sublevels determined respectively by flat and hierarchical nominative data.

Three kinds of logics can be constructed from program models on the flat nominative data level:

1) pure quasiary predicate logics based on algebras with one sort: $Pr^{V,A}$;
2) quasiary predicate-function logics based on algebras with two sorts: $Pr^{V,A}$ and $Fn^{V,A}$;
3) quasiary program logics based on algebras with three sorts: $Pr^{V,A}$, $Fn^{V,A}$, and $Prg^{V,A}$ (will be considered in the next chapter).

For logics of pure quasiary predicates (*pure* CNL) we identify renominative, quantifier, and quantifier-equational levels.

*Renominative* logics [NikSh2008, NikSh2013] are the most abstract among above-mentioned logics. The main new composition for these logics is the composition of renomination (renaming) $R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n} : Pr^{V,A} \xrightarrow{t} Pr^{V,A}$. Intuitively, given a quasiary predicate $p$ and a nominative set $d$ the value of $R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n}(p)(d)$ is evaluated in the following way: first, a new nominative set $d'$ is constructed from $d$ by changing the values of the names $v_1,\ldots,v_n$ in $d$ to the values of the names $x_1,\ldots, x_n$ respectively; then the predicate $p$ is applied to $d'$. The obtained value (if it was evaluated) will be the result of $R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n}(p)(d)$. For simplicity's sake we will also use simplified nota-

tion $R_{\bar{x}}^{\bar{v}}$ for renomination composition. The basic composition operations of renominative logics are $\vee$, $\neg$, and $R_{\bar{x}}^{\bar{v}}$.

At the *quantifier* level, all basic values can be used to construct different nominative sets to which quasiary predicates can be applied. This allows one to introduce the compositions of quantification $\exists x$ in style of Kleene's strong quantifiers. The basic compositions of logics of the quantifier level are $\vee$, $\neg$, $R_{\bar{x}}^{\bar{v}}$, and $\exists x$.

At the *quantifier-equational* level, new possibilities arise for equating and differentiating values with special 0-ary compositions, i.e., equality predicates $=_{xy}$. Basic compositions of logics of the quantifier-equational level are $\vee$, $\neg$, $R_{\bar{x}}^{\bar{v}}$, $\exists x$, and $=_{xy}$.

All specified logics (renominative, quantifier, and quantifier-equational) are based on algebras that have only one sort: a class of quasiary predicates.

For quasiary predicate-function logics we identify the function level and the function-equational levels.

At the *function* level, we have extended capabilities of formation of new arguments for functions and predicates. In this case it is possible to introduce the superposition compositions $S^{\bar{x}}$ (see [NikSh2013]), which formalizes substitution of functions into predicate. It seems also natural to introduce special 0-ary compositions, called denomination (denaming) functions $'x$. They can be considered a parametric unary function with one parameter $x$. Given a nominative set, $'x$ yields a value of the name $x$ in this set. Introduction of such functions allows one to model renomination compositions with the help of superposition. The basic compositions of logics at the function level are $\vee$, $\neg$, $S^{\bar{x}}$, $\exists x$, and $'x$.

At the function-equational level, a special equality composition = can be introduced additionally [23]. The basic compositions of logics of the function-equational level are $\vee$, $\neg$, $S^{\bar{x}}$, $\exists x$, $'x$, and =. At this level different classes of first-order logics can be presented.

This means that two-sorted algebras (with sets of predicates and functions as sorts and above-mentioned compositions as operations) form a semantic base for first-order CNL.

To preserve properties of classical first-order logic in first-order CNL we should restrict the class $^{V}A \xrightarrow{p} Bool$ of quasiary predicates. Namely, we introduce a class of equitone predicates and its different variations such as maxitotal equitone, equicompatible, etc. [NikSh2008,NikSh2013]. A predicate $p$: $^{V}A \xrightarrow{p} Bool$ is called *equitone* if for every $d$, $d' \in {}^{V}A$ such that $d \subseteq d'$ from $p(d)\downarrow = b$ follows that $p(d')\downarrow = b$; if an equitone predicate $p$ is defined on all elements of $A^{V}$ then $p$ is called *maxitotal equitone*; if a predicate $p$ is a restriction of some equitone predicate then $p$ is *equicompatible* predicate. Logics based on maxitotal equitone, equitone, and equicompatible predicates are the "closest" generalization of the classical first-order logic that preserves its main properties. These logics are called *neoclassical logics* [NikSh2008].

### *3.2. Composition-nominative logics of quasiary predicates*

Here we consider only pure CNL of the following abstraction levels: *propositional*, *renominative*, and *quantifier*. Each level specifies classes of predicates, defined on classes of data, and a set of compositions that correspond to this level. This means that a class of predicate algebras should be specified for each level. Such algebras form the semantic base for CNL of chosen level. Predicate symbols are interpreted as predicates, and formulas are considered as terms, interpreted in such algebras.

Let $Bool=\{F,T\}$ be a set of Boolean (logical) values, $Pr(Dt)=Dt \xrightarrow{p} Bool$ be the set of all partial predicates specified on some data set $Dt$, $C$ be a set of total $n$-ary compositions of predicates of the type $Pr(Dt)^{n} \xrightarrow{t} Pr(Dt)$. Then a pair $<Pr(Dt), C>$ is *an algebra of partial predicates*. In terms of traditional logic, the set $Dt$ is interpreted as a possible world or universe, elements of $Dt$ are treated as world states, predicates from $Pr(Dt)$ are considered as world

state properties, and compositions from $C$ are treated as operators over $Pr(Dt)$.

Parameters of logics (classes of predicates, interpretations of predicate symbols, and languages) can be restricted.

Examples of restricted predicate classes are total predicates, equitone predicates, etc. Interpretations can also be restricted, for example some predicate symbols in the language can have a fixed interpretation. A restricted language can consist only of formulas that are in some special normal form. Thus, we will obtain logics of total predicates, logics with fixed interpretations, logics with restricted classes of formulas. Such logics are called restricted.

These considerations lead to the following definitions of the notion of a concrete (especial) logic.

A concrete composition-nominative logic $Lg$ is specified by

- a signature ($Cs$, $Ps$), where $Cs$ is a set of composition symbols and $Ps$ is a set of predicate symbols;
- a certain class of data sets; an element on this class is usually denoted by $Dt$;
- a certain class of algebras of the form $<Prr, C>$, where $Prr \subseteq Pr(Dt)$, $Dt$ is one of data sets from the class specified for $Lg$, and operations (compositions) $C$ are interpretations of $Cs$ in this algebra;
- a certain class of interpretations of predicate symbols of the form $I^{Ps}: Ps \xrightarrow{t} Prr$ for each algebra $<Prr, C>$;
- a language of $Lg$ , which is a subset of all formulas of the signature ($Cs$, $Ps$), constructed over predicate symbols from the set $Ps$ with the help of symbols of compositions from the set $Cs$. The class of all formulas $Fr_{Lg}(Cs, Ps)$ can be defined inductively:

  1) if $P \in Ps$ then $P \in Fr_{Lg}(Cs, Ps)$ ;
  2) if $\Phi_1, \ldots, \Phi_n \in Fr_{Lg}(Cs, Ps)$ , $c \in Cs$ is an $n$-ary composition symbol then $c(\Phi_1, \ldots, \Phi_n) \in Fr_{Lg}(Cs, Ps)$.

Note, that for simplicity's sake we will use the same notation for compositions as algebraic operations (the set *C*) and their symbols in the language signature (the set *Cs*). The class $Fr_{Lg}(Cs, Ps)$ is exactly the class of all terms of algebras specified for *Lg*. In the sequel we consider formulas of $Fr_{Lg}(Cs, Ps)$ in their traditional form using infix operations and brackets; brackets can be omitted according to common rules for the priorities of operations (priority of binary disjunction is weaker than priority of unary operations).

From these definitions follows that the main semantic object in a CNL *Lg* is a class of algebras of the form $<Prr, C>$, where $Prr \subseteq Pr(Dt)$ for some *Dt*. In CNL compositions have a fixed interpretation. Usually, the set of compositions is clear from the context, therefore we will represent such algebras by a pair (*Dt*, *Prr*), which is called α-*interpretation*. Interpretations of predicate symbols of the form $I^{Ps}:Ps \xrightarrow{t} Prr$ in an algebra $<Prr, C>$ will be called σ-*interpretations*. If *Ps* is clear from the context we write *I*. A combination of an α-interpretation and a σ-interpretation gives a π-*interpretation* denoted $(Dt, Prr, I^{Ps})$. Such interpretations may also be called models of a logic. Given a π-interpretation $J=(Dt, Prr, I^{Ps})$ any formula $\Phi \in Fr_{Lg}(Cs, Ps)$ by usual procedure of term interpretation can be interpreted as a certain predicate denoted $\Phi_J \in Prr$. Prefixes π-, α-, σ- may be omitted.

Now we will give formal definitions for three types of composition-nominative logics considered at propositional, renominative, and quantifier levels. Logics of such types will be denoted as PCNL, RCNL, and QCNL respectively. We will consider general (unrestricted) pure CNL (without function symbols).

In definitions we will use the following notation:

- $p(d)\downarrow= b$ means that a predicate *p* is defined on data *d* with a Boolean value *b*;
- $p(d)\uparrow$ means that a predicate *p* on *d* is undefined.

**Composition-nominative logic of propositional level**

Logics of propositional level are called *propositional composition-nominative logics* (PCNL). An unrestricted concrete logic $L_P$ of this level is determined by two basic compositions: disjunction $\vee$ and negation $\neg$. Thus, the signature of $L_P$ is $(\{\vee, \neg\}, Ps)$. Any data set $Dt$ belongs to the class of data sets specified for $L_P$, thus semantic base of $L_P$ is the class of algebras of the form

$$AP(Dt) = \langle Pr(Dt), \vee, \neg \rangle.$$

Basic compositions are Kleene's disjunction $\vee$ and negation $\neg$, defined by the following formulas ($p, q \in Pr(Dt)$, $d \in Dt$):

$$(p \vee q)(d) = \begin{cases} T, \text{ if } p(d)\downarrow = T \text{ or } q(d)\downarrow = T, \\ F, \text{ if } p(d)\downarrow = F \text{ and } q(d)\downarrow = F, \\ \text{undefined in other cases.} \end{cases}$$

$$(\neg p)(d) = \begin{cases} T, \text{if } p(d)\downarrow = F, \\ F, \text{if } p(d)\downarrow = T, \\ \text{undefined if } p(d)\uparrow. \end{cases}$$

Language of $L_P$, represented by the class of formulas

$$Fr_{PCNL}(\{\vee, \neg\}, Ps),$$

is defined inductively:

1. If $P \in Ps$ then $P \in Fr_{PCNL}(\{\vee, \neg\}, Ps)$. Such formula is called an atomic formula.

2. If $\Phi, \Psi \in Fr_{PCNL}(\{\vee, \neg\}, Ps)$ then $(\Phi \vee \Psi) \in Fr_{PCNL}(\{\vee, \neg\}, Ps)$ and $\neg\Phi \in Fr_{PCNL}(\{\vee, \neg\}, Ps)$.

**Composition-nominative logic of renominative level**

Logics of renominative level are called *renominative composition-nominative logics* (RCNL). An unrestricted concrete logic $L_R$ of this level is determined by three basic compositions: disjunction $\vee$, negation $\neg$, and renomination $R_{\bar{x}}^{\bar{v}} : Pr(^V A) \xrightarrow{t} Pr(^V A)$, where $\bar{v} = (v_1, ..., v_n)$ and $\bar{x} = (x_1, ..., x_n)$ are lists of variables from a cer-

tain set $V$, fixed for $L_R$; variables from $\bar{v}$ are called *upper names of renomination composition* and should be distinct, variables from $\bar{x}$ are called *lower names of renomination composition*, $n \geq 0$. Thus, the signature of $L_R$ is $(\{\vee, \neg, R_{\bar{x}}^{\bar{v}}\}, Ps)$. Please note that $R_{\bar{x}}^{\bar{v}}$ is a parametric composition, which represents a class of renomination compositions with different parameters constructed from elements of $V$. It means that the set of composition symbols actually is $\{\vee, \neg\} \cup \{ R_{\bar{x}}^{\bar{v}} \mid \bar{x}, \bar{y} \in V^* \text{ and } \bar{x}, \bar{y} \text{ satisfy the restrictions on parameters of renomination composition}\}$. This notational convention concerns signatures of other logics with parametric compositions. For every set $A$ the set ${}^VA$ of all nominative sets belongs to the class of data sets specified for $L_R$. Semantic base of $L_R$ are all algebras of the form

$$AR(V, A) = <Pr({}^VA), \vee, \neg, R_{\bar{x}}^{\bar{v}}>.$$

The set $Pr({}^VA)$ will also be denoted $Pr^{V,A}$.

Unary parametric composition of *renomination* $R_{x_1,\ldots,x_n}^{v_1,\ldots,v_n}$: $Pr^{V,A} \xrightarrow{t} Pr^{V,A}$ is defined by the following formula ($p \in Pr^{V,A}$, $d \in {}^VA$):

$$(R_{x_1,\ldots,x_n}^{v_1,\ldots,v_n}\ p)\,(d) =$$
$$= p([v \mapsto a \in_n d \mid v \notin \{v_1,\ldots,v_n\}] \nabla [v_i \mapsto d(x_i) \mid d(x_i)\downarrow, i \in \{1,\ldots,n\}]).$$

The $\nabla$ operation is defined as follows. If $d_1$ and $d_2$ are two nominative sets, then $d = d_1 \nabla d_2$ consists of all named pairs of $d_2$ and only those pairs of $d_1$, whose names are not defined in $d_2$. Note that $R\,p$ (when $n = 0$) is equal to $p$.

Language of $L_R$ is represented by the class of formulas

$$Fr_{RCNL}(\{\vee, \neg, R_{\bar{x}}^{\bar{v}}\}, Ps).$$

To simplify notation, we write $Fr_{RCNL}(V, Ps)$. This class is defined inductively by rules 1 and 2 from the definition of $Fr_{PCNL}(\{\vee, \neg\}, Ps)$ in which $Fr_{PCNL}(\{\vee, \neg\}, Ps)$ is changed on $Fr_{RCNL}(V, Ps)$ and by the following new rule:

3. If $\bar{v} = (v_1,...,v_n)$, $\bar{x} = (x_1,...,x_n)$, $\bar{v}$ is a list of distinct variables, $v_i$, $x_i \in V$ for all $i \in \{1,...,n\}$ $(n \geq 0)$, $\Phi \in Fr_{RCNL}(V, Ps)$ then $R^{\bar{v}}_{\bar{x}}\Phi \in Fr_{RCNL}(V, Ps)$.

**Composition-nominative logic of quantifier level**

Logics of quantifier level are called *quantifier composition-nominative logics* (QCNL). An unrestricted concrete logic $L_Q$ of this level is determined by four basic compositions: disjunction $\vee$, negation $\neg$, renomination $R^{\bar{v}}_{\bar{x}}$, and existential quantification $\exists x$, where $x \in V$, $\bar{x}, \bar{v}$ are vectors of variables from $V$. Thus, the signature of $L_R$ is ($V$, $\{\vee, \neg, R^{\bar{v}}_{\bar{x}}, \exists x\}$, $Ps$). Let us note that elements from the set of names $V$ are used as parameters of existential quantification (and renomination too).

The class of data sets specified for $L_Q$ is the same as one specified for $L_R$. Semantic base of $L_Q$ are all algebras of the form

$$AQ(V, A) = \langle Pr^{V,A}, \vee, \neg, R^{\bar{v}}_{\bar{x}}, \exists x \rangle.$$

At the quantifier level predicates can be applied to all data obtained from given data by changing the basic values of the component with a fixed name. Therefore, a composition of existential quantification $\exists x$ with the parameter $x \in V$ is defined by the following formula ($p \in Pr^{V,A}$, $d \in Dt$):

$$(\exists x\ p)(d) = \begin{cases} T, \text{ if } b \in A \text{ exists}: p(d\nabla x \mapsto b)\downarrow = T, \\ F, \text{ if } p(d\nabla x \mapsto a)\downarrow = F \text{ for each } a \in A, \\ \text{undefined in other cases.} \end{cases}$$

Here $d\nabla x \mapsto a$ is a shorter form for $d\nabla[x \mapsto a]$. Language of $L_Q$, represented by the class of formulas $Fr_{QCNL}(V, Ps)$, is defined inductively by rules 1–3 from the definition of $Fr_{RCNL}(V, Ps)$ in which $Fr_{RCNL}(V, Ps)$ is changed to $Fr_{QCNL}(V, Ps)$ and by the following new rule:

4. If $x \in V$, $\Phi \in Fr_{QCNL}(V, Ps)$ then $\exists x\Phi \in Fr_{QCNL}(V, Ps)$.

Let us note that for all described logics derived compositions (such as conjunction &, universal quantification $\forall x$, etc.) are defined in a traditional way.

Here we develop reductive methods for solving satisfiability problem. It means that we reduce this problem to the satisfiability problem in classical logics. So, we will also be referring to several fragments of classical first-order logic. We consider the pure first-order logic, that is, classical first-order logic without function symbols.

Now we give definitions of three classical logics: propositional logic, quantifier-free predicate logic, and first-order predicate logic. We will give their definitions in the style of CNL describing only distinctions from corresponding CNL.

**Classical propositional logic**

Logics of propositional level are called *classical propositional logics* (PCL). A concrete logic $L_{PCL}$ of this level is specified by the signature $(\{\vee, \neg\}, Ps)$. A class of data sets consists only of one empty set $\varnothing$. In this case we should specify a class of predicates $\varnothing \xrightarrow{t} Bool$. For simplicity's sake we identify $\varnothing \xrightarrow{t} Bool$ with *Bool*. This is a disputable solution but we follow a mathematical tradition of identifying null-ary predicates with Boolean constants. Of course, we should ensure that all definitions agree with this solution. Accordingly, definitions of compositions should also take this fact into account. This solution gives us only one algebra $APCL = \langle Bool, \vee, \neg \rangle$ and only one class of $\sigma$-interpretations of the form $I^{Ps}{}_{PCL}$: $Ps \xrightarrow{t} Bool$. The language $Fr_{PCL}(\{\vee, \neg\}, Ps)$ is the same as $Fr_{PCNL}(\{\vee, \neg\}, Ps)$.

**Classical quantifier-free predicate logic**

Logics of quantifier-free level are called here *classical quantifier-free logics* (FCL). A concrete logic $L_{FCL}$ of this level is specified by the signature $(V, \{\vee, \neg\}, Ps, arity)$ where $V$ is a set of variables,

and *arity*: $Ps \xrightarrow{t} \{0,1,2, \ldots\}$ is a function that for each predicate symbol yields its arity.

For every set $A$ the set $A^V = V \xrightarrow{t} A$ of all total nominative sets belongs to the class of data sets specified for $L_{FCL}$. A class of total predicates $TPr(A^V) = A^V \xrightarrow{t} Bool$ is used for interpretation of formulas. The semantic base of FCL are algebras of the form

$$AFCL(V,A) = \langle TPr(A^V), \vee, \neg \rangle.$$

Interpretations of predicate symbols are also defined differently from those in RCNL. They are mappings

$$I^{Ps}_{CL}: Ps \xrightarrow{t} \bigcup_{n \geq 0} (A^n \xrightarrow{t} Bool)$$

such that $I^{Ps}_{CL}(P) \in A^n \xrightarrow{t} Bool$ if $arity(P) = n$ for $P \in Ps$. Thus, interpretations have the form $K^{Ps}_{CL} = (A^V, TPr(A^V), I^{Ps}_{CL})$. Such interpetations will be called κ-interpetations.

The language $Fr_{FCL}(V, Ps)$ is defined inductively:

1. If $P \in Ps$, $arity(P) = n$, and $x_1, \ldots, x_n \in V$, then $P(x_1, \ldots, x_n) \in Fr_{FCL}(V, Ps)$. Such formula is called an atomic formula.
2. If $\Phi, \Psi \in Fr_{FCL}(V, Ps)$ then $(\Phi \vee \Psi) \in Fr_{FCL}(V, Ps)$ and $\neg\Phi \in Fr_{FCL}(V, Ps)$.

A κ-interpretation $K^{Ps}_{CL} = (A^V, TPr(A^V), I^{Ps}_{CL})$, in a simplified form denoted by $K$, for every atomic formula $P(x_1, \ldots, x_n)$ defines its meaning in $TPr(A^V)$ as a predicate $P(x_1, \ldots, x_n)_K$ such that $P(x_1, \ldots, x_n)_K(d) = I^{Ps}_{CL}(P)(d(x_1), \ldots, d(x_n))$ for every $d \in A^V$. The meaning $\Phi_K$ of any formula $\Phi \in Fr_{FCL}(V, Ps)$ is defined in a usual way.

**Classical First-Order Predicate Logic**

Logics of quantifier level are called *classical first-order predicate logics* (QCL). A concrete logic $L_{QCL}$ of this level is specified in the same way as $L_{FCL}$ with such distinctions: the signature is

$$(V, \{\vee, \neg, \exists x\}, Ps, arity);$$

considered algebras are $AQCL = \langle TPr(A^V), \vee, \neg, \exists x \rangle$; for defining $Fr_{QCL}(V, Ps)$ we use an additional rule

3. If $x \in V$, $\Phi \in Fr_{QCL}(Ps)$ then $\exists x\Phi \in Fr_{QCL}(V, Ps)$.

The logics PCL, FCL, and QCL correspond to PCNL, RCNL, and QCNL respectively.

### *3.3. Satisfiability problem for composition-nominative logics*

The satisfiability problem is one of the classical problems in logic [Mendelson1966]. Lately, the interest to this problem has increased due to practical value it has obtained in such areas as program verification, synthesis, analysis, testing etc. [Kroening2008, Marques-Silva2008, Nieuwenhuis2006]. Here we address the satisfiability problem in the context of the composition-nominative approach [Nik1998], which aims to construct a hierarchy of logics of various abstraction and generality levels on the methodological basis, which is common with programming. The main principles of the approach are principles of development from abstract to concrete, priority of semantics, compositionality, and nominativity.

Let $\Phi \in Fr_{Lg}(Cs, Ps)$. We assume that in the algebras of the form <*Prr*, *C*> compositions from *C* are interpretations of composition symbols *Cs*. We also assume that a set of predicate symbols *Ps* is fixed throughout this section. Formula Φ is called *satisfiable in a π-interpretation* $J = (Dt, Prr, I)$ if there is a $d \in Dt$ such that $\Phi_J(d)\downarrow = T$. We shall denote this by $J \models \Phi$ (also notation $J \models\sim \Phi$ is used). Let us note that in classical predicate logic *d* is called *variable valuation* or *variable assignment*. A CNL formula Φ is called *satisfiable in a predicate class Prr* (we write $(Dt, Prr) \models \Phi$) if there exists an interpretation $J = (Dt, Prr, I)$ in which Φ is satisfiable. A CNL formula Φ is called *satisfiable* if there exists an interpretation *J* in which Φ is satisfiable. We shall denote this as $\models \Phi$. We call formulas Φ and Ψ *equisatisfiable* if they are either both satisfiable or both not satisfiable (i.e. unsatisfiable).

Satisfiability of a formula is related to its validity. A CNL formula $\Phi$ is called *valid in a $\pi$-interpretation* $J = (Dt, Prr, I)$ if there is no $d \in Dt$ such that $\Phi_J(d)\downarrow = F$. We shall denote this as $J \models \Phi$, which means that $\Phi$ is not refutable in $J$. Such definition of validity is chosen in accordance with the principle of the ascent (development) from the abstract to the concrete. This is the simplest (the most abstract) treatment of validity, which specifies only two cases: "valid" and "not valid". A CNL formula $\Phi$ is called *valid in a predicate class Prr* (we write $(Dt, Prr) \models \Phi$) if for every interpretation $J = (Dt, Prr, I)$ we have $J \models \Phi$. Formula $\Phi$ is called *valid* if $J \models \Phi$ for every interpretation $J$. We call formulas $\Phi$ and $\Psi$ *equivalent* if $\Phi_J = \Psi_J$ for every interpretation $J$.

If we need to emphasize that a formula $\Phi$ belongs to the language of a specific logic (e.g. PCNL, QCNL, FCL, etc.), we will use the corresponding subscript: (e.g. $\Phi_{PCNL}$, $\Phi_{QCNL}$, $\Phi_{FCL}$ etc.) When needed we will include the corresponding logic abbreviation in the satisfiability sign $|\approx$, e.g. $|\approx_{PCNL}, |\approx_{QCNL}, |\approx_{FCL}$, etc.

Validity and satisfiability are related with each other. But due to possible presence of a nowhere defined predicate (which is a valid predicate) we do not have in CNL the property that $\Phi$ is satisfiable if $\Phi$ is valid (which holds for classical first-order logic). But reduction of satisfiability to validity still holds in CNL. We formulate this statement for every concrete CNL *Lg* .

**Lemma 3.1.**

Let $\Phi \in Fr_{Lg}(Cs, Ps)$ and $<Pr(Dt), C>$ be an algebra of partial predicates, $J=(Dt, Pr(Dt), I)$ be a $\pi$-interpretation. Formula $\Phi$ is satisfiable in $J$ iff $\neg\Phi$ is not valid in $J$.

Consider an algebra $AP(Dt)=<Pr(Dt), C>$. We say that $p \subseteq q$, where $p, q \in Pr(Dt)$, if for every $d \in Dt$ such that $p(d)\downarrow = b$ we have $q(d)\downarrow = b$ (this means inclusion of predicate graphs). We say

that $(p_1,\ldots,p_n) \subseteq (q_1,\ldots,q_n)$ if $p_i \subseteq q_i$ for every $i \in \{1,...,n\}$. An $n$-ary composition $c \in C$ is called *monotone* if

from $(p_1,\ldots,p_n) \subseteq (q_1,\ldots,q_n)$ follows $c(p_1,\ldots,p_n) \subseteq c(q_1,\ldots,q_n)$.

**Lemma 3.2.**

Compositions in algebras of the forms $AP(Dt) = <Pr(Dt), \vee, \neg>$, $AR(V, A) = <Pr^{V,A}, \vee, \neg, R^{\bar{v}}_{\bar{x}}>$, and $AQ(V, A) = <Pr^{V,A}, \vee, \neg, R^{\bar{v}}_{\bar{x}}, \exists x>$ are monotone.

*Proof.*

The proof is straightforward. For example, let us show monotonicity of $\exists x$. Assume $p, q \in Pr^{V,A}$, $p \subseteq q$. Let $\exists x\, p(d)\downarrow = T$. Then there exists $b \in A$ such that $p(d\nabla x \mapsto b)\downarrow = T$; thus, we have that $q(d\nabla x \mapsto b)\downarrow = T$; therefore $\exists x\, q(d)\downarrow = T$. Let $\exists x\, p(d)\downarrow = F$. Then $p(d\nabla x \mapsto a)\downarrow = F$ for every $a \in A$; thus, we have that $q(d\nabla x \mapsto a)\downarrow = F$ for every $a \in A$. This means that $\exists x\, q(d)\downarrow = F$.

Monotonicity of CNL compositions guarantees stability of satisfiability under predicate extensions.

Let $J_1 = (Dt, Pr(Dt), I_1)$, $J_2 = (Dt, Pr(Dt), I_2)$ be two interpretations such that $I_1(P) \subseteq I_2(P)$ for every $P \in Ps$. Then we say that $J_2$ extends $J_1$ and denote this by $J_1 \leq J_2$.

**Lemma 3.3.**

Let $\Phi \in Fr_{Lg}(Cs, Ps)$ and $<Prr, C>$ be an algebra of partial predicates. Let $J_1 = (Dt, Prr, I_1)$, $J_2 = (Dt, Prr, I_2)$ be two interpretations such that $J_1 \leq J_2$. Suppose that all compositions in $C$ are monotone. Then from $J_1 |\approx \Phi$ follows $J_2 |\approx \Phi$.

The statement can easily be proved by induction over the structure of formula $\Phi$.

From Lemma 3.3 follows that given a $\pi$-interpretation $J = (Dt, Prr, I)$ such that $J |\approx \Phi$ we can construct an interpretation $K = (Dt, TPr(Dt), I')$ within the class of total predicates such that $J \leq K$ and $K |\approx \Phi$.

**Lemma 3.4.**

Let $\Phi \in Fr_{Lg}(Cs, Ps)$. Then from $(Dt, Prr) \mid\approx \Phi$ follows $(Dt, TPr(Dt)) \mid\approx \Phi$.

For all three CNL levels considered here the classes of total predicates are closed under compositions of respective levels.

**Lemma 3.5.**

The class $TPr(Dt) = Dt \xrightarrow{t} Bool$ of total predicates over $Dt$ forms a sub-algebra in $AP(Dt)$; the class $TPr({}^{V}A) = ({}^{V}A \xrightarrow{t} Bool)$ is a sub-algebra in $AR(V, A)$ and in $AQ(V, A)$.

**Satisfiability for propositional CNL**

Let $TCPr(Dt) \subseteq TPr(Dt)$ be a set of total constant predicates $TCPr(Dt) = \{Tp, Fp\}$ such that $Tp(d)\downarrow = T$ and $Fp(d)\downarrow = F$ for every $d \in Dt$.

**Lemma 3.6.**

Let $\Phi \in Fr_{PCNL}(\{\vee, \neg\}, Ps)$, $(Dt, TPr(Dt))$ be an $\alpha$-interpretation. Then $(Dt, TPr(Dt)) \mid\approx \Phi$ iff $(Dt, TCPr(Dt)) \mid\approx \Phi$.

*Proof.*

To prove this lemma, we need to show that for every $\pi$-interpretation $J = (Dt, TPr(Dt), I)$ such that $J \mid\approx \Phi$ we can construct a $\pi$-interpretation $K = (Dt, TCPr(Dt), I')$ such that $K \mid\approx \Phi$. From $J \mid\approx \Phi$ it follows that there exists $d \in Dt$ such that $\Phi_J(d)\downarrow = T$. For every $P \in Ps$ let $I'(P)$ be a total constant predicate taking the value $I(P)(d)$. It can easily be shown that $\Phi_K(d) \downarrow = T$, which means that $K \mid\approx \Phi$. The inverse direction holds because $TCPr(Dt) \subseteq TPr(Dt)$.

Lemma 3.6 justifies considering only $\sigma$-interpretations in the set of total constant predicates for checking satisfiability of propositional CNL formulas. In fact, such reductions can be continued by con-

sidering an empty data set. In this case $TCPr(\varnothing)=Bool$ as was discussed earlier.

**Lemma 3.7.**

Let $\Phi \in Fr_{PCNL}(\{\vee, \neg\}, Ps)$, $d \in Dt$. Then $(Dt, TCPr(Dt)) \mid\approx \Phi$ iff $(\varnothing, TCPr(\varnothing)) \mid\approx \Phi$.

The proof is trivial.

Summarizing formulated results, we obtain the following chain of reductions:

$$(Dt,Pr(Dt)) \mid\approx \Phi \Leftrightarrow (Dt, TPr(Dt)) \mid\approx \Phi \Leftrightarrow$$
$$\Leftrightarrow (Dt, TCPr(Dt)) \mid\approx \Phi \Leftrightarrow (\varnothing, TCPr(\varnothing)) \mid\approx \Phi.$$

This essentially means that for checking satisfiability of $\Phi$ in CNL of the propositional level we can use the methods for checking satisfiability in the classical propositional logic. Thus, we have proved the following theorem.

**Theorem 3.1.**

Let $\Phi \in Fr_{PCNL}(\{\vee, \neg\}, Ps)$. Then $\mid\approx_{PCNL} \Phi$ iff $\mid\approx_{PCL} \Phi$.

For checking satisfiability in classical propositional logic many methods were developed. Among them we would like to mention Davis-Putnam-Logemann-Loveland algorithm [Davis1952], variations of which are very widely used.

**Satisfiability problem for renominative CNL**

In this section we suggest a technique that allows reducing the satisfiability problem for CNL formulas of the renominative level to the satisfiability problem for quantifier-free formulas of classical predicate logic. First, we put an RCNL formula into a special equivalent representation in which renomination composition is applied only to predicate symbols, then we strengthen this representation to a unified renominative form, and at last, we apply syntactical trans-

formation rules that construct an equisatisfiable formula of classical quantifier-free predicate logic.

A (sub)formula of the form $R^{\bar{v}}_{\bar{x}} P$ where $P \in Ps$ is called *a renominative atom*. An RCNL formula $\Phi$ is said to be in *renominative normal form* (RNF) if renomination compositions and predicate symbols occur in $\Phi$ only in renominative atoms. Recall that lists of names in renomination compositions may be empty (empty renomination). In this case renominative atoms have the form $R\,P$ $(P \in Ps)$. Here we assume that a non-empty renomination occurs in $\Phi$; otherwise $\Phi$ can be trivially reduced to a propositional formula.

We consider a total mapping $rnf : Fr_{RCNL}(V, Ps) \xrightarrow{t} Fr_{RCNL}(V, Ps)$ that maps an RCNL formula to its RNF. Given an RCNL formula $\Phi$, renominative normal form $rnf[\Phi]$ is constructed as described by rules R1–R7:

R1) $rnf[P] = R\,P$ $(P \in Ps)$

R2) $rnf[(\Phi_1 \vee \Phi_2)] = (rnf[\Phi_1] \vee rnf[\Phi_2])$

R3) $rnf[\neg\Phi] = \neg rnf[\Phi]$

R4) $rnf[R^{\bar{v}}_{\bar{x}} P] = R^{\bar{v}}_{\bar{x}} P$ $(P \in Ps)$

R5) $rnf[R^{\bar{v}}_{\bar{x}} (\Phi_1 \vee \Phi_2)] = (rnf[R^{\bar{v}}_{\bar{x}} \Phi_1] \vee rnf[R^{\bar{v}}_{\bar{x}} \Phi_2])$

R6) $rnf[R^{\bar{v}}_{\bar{x}} \neg\Phi] = \neg rnf[R^{\bar{v}}_{\bar{x}} \Phi]$

R7) $rnf[R^{v_1,\ldots,v_n,w_1,\ldots,w_m}_{x_1,\ldots,x_n,y_1,\ldots,y_m}\ R^{v_1,\ldots v_i,u_1,\ldots,u_k}_{s_1,\ldots,s_i,z_1,\ldots,z_k} \Phi] = rnf[R^{v_1,\ldots,v_n,w_1,\ldots,w_m,u_1,\ldots,u_k}_{\alpha_1,\ldots,\alpha_n,y_1,\ldots,y_m,\beta_1,\ldots,\beta_k} \Phi]$,

where

$w_i \neq u_j$ $(i = 1,\ldots,m;\ j = 1,\ldots,k)$, $\alpha_i = s_i(v_1,\ldots,v_n,w_1,\ldots,w_m / x_1,\ldots,x_n, y_1,\ldots,y_m)$, $\beta_j = z_j(v_1,\ldots,v_n, w_1,\ldots,w_m / x_1,\ldots,x_n, y_1,\ldots,y_m)$. Here $r(b_1,\ldots,b_q / c_1,\ldots,c_q) = r$ if $r \notin \{b_1,\ldots,b_q\}$, $r(b_1,\ldots,b_q / c_1,\ldots,c_q) = c_i$ if $r = b_i$ for some $i$. This rule represents explicitly the result of function composition of parameters of two successive renominations.

The transformation $rnf : Fr_{RCNL}(V, Ps) \xrightarrow{t} Fr_{RCNL}(V, Ps)$ relies on equivalent formula transformations of RCNL[NikSh2008], thus the following lemma holds.

**Lemma 3.8.**

Let $\Phi \in Fr_{RCNL}(V, Ps)$. Then $|\approx \Phi$ iff $|\approx rnf[\Phi]$.

Note that proposed way of construction of renominative normal form preserves the reciprocal order of propositional compositions in the original formula. Thus, the obtained formula is practically of the same complexity as the initial formula.

An RCNL formula $\Phi$ is said to be in a *unified renominative normal form* (URNF) if it is in renominative normal form and for every pair of its renominative atoms $R\frac{\bar{u}}{\bar{q}}P$ and $R\frac{\bar{w}}{\bar{y}}Q$ we have that vectors $\bar{u}$ and $\bar{w}$ coincide; that is, in all renominative atoms the lists of their upper names are the same.

We will define a total multi-valued (non-deterministic) mapping $urnf : Fr_{RCNL} \xrightarrow{tm} Fr_{RCNL}$ that transforms an RCNL formula to its URNF. It means that given an RCNL formula $\Phi$ we non-deterministically construct $urnf[\Phi]$.

Let $\Phi$ be an arbitrary RCNL formula in renominative normal form. Denote by $V_\Phi$ the set of all names that occur as upper names in renominative atoms of $\Phi$. To construct URNF we unify all renominative atoms as follows. First, we choose an ordering of variables (names) from $V_\Phi$ and get a list $\bar{v}$; then we transform all renominative atoms of the form $R\frac{\bar{u}}{\bar{q}}P$ occurring in $\Phi$ to the form $R\frac{\bar{v}}{\bar{x}}P$, so that $\bar{v}$ becomes the list of upper names in renomination composition, while the lists of lower names may differ. This is done using the *identical renomination rule*: $R\frac{\bar{u}}{\bar{q}}P = R\frac{z,\bar{u}}{z,\bar{q}}P$, and *commutative renomination rule*: $R^{u_1,\ldots,u_i,\ldots,u_j,\ldots,u_n}_{q_1,\ldots,q_i,\ldots,q_j,\ldots,q_n}P = R^{u_1,\ldots,u_j,\ldots,u_i,\ldots,u_n}_{q_1,\ldots,q_j,\ldots,q_i,\ldots,q_n}P$. For a formula $\Phi$ we will denote any possible URNF of $\Phi$ by *urnf* $[\Phi]$.

**Lemma 3.9.**

Let $\Phi \in Fr_{RCNL}(V, Ps)$. Then $|\approx \Phi$ iff $|\approx urnf[\Phi]$.

*Proof.*

Applications of identical and commutative renomination rules preserve equivalence of formulas, therefore the formulas $\Phi$ and *urnf*[$\Phi$] are equivalent. Consequently, they are equisatisfiable.

Now we describe a transformation

$$clf : Fr_{RCNL}(V, Ps) \xrightarrow{p} Fr_{FCL}(V, Ps)$$

which given an RCNL formula in URNF yields a formula of quantifier-free classical logic as follows:

1. $clf[R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n} P] = P(x_1,\ldots,x_n)$
2. $clf[(\Phi_1 \vee \Phi_2)] = (clf[\Phi_1] \vee clf[\Phi_2])$
3. $clf[\neg\Phi] = \neg clf[\Phi]$

Transformation *clf* preserves satisfiability.

Before proceeding to the proof we introduce several mappings over nominative sets and predicates related with a special constant $\varepsilon$. This constant represents the case when a value of a variable is not defined in a nominative set.

Let $A_\varepsilon = A \cup \{\varepsilon\}, \varepsilon \notin A$. First, define $\varepsilon$-adding transformation ($\varepsilon$-totalizing) *ade*: ${}^V A \xrightarrow{t} A_\varepsilon{}^V$ by the formula

$$ade(d) = [v \mapsto \varepsilon \mid \text{for each } v \in V] \nabla d.$$

In other words

$$ade(d) = [v \mapsto d(v) \mid v \in V, d(v)\downarrow] \nabla [v \mapsto \varepsilon \mid v \in V, d(v)\uparrow].$$

It is clear that *ade* is bijective. The inverse mapping will be denoted *dee*: $A_\varepsilon{}^V \xrightarrow{t} {}^V A$. Formally,

$$dee(d) = [v \mapsto d(v) \mid v \in V, d(v) \neq \varepsilon].$$

In the same way we define two bijective mappings between classes of total predicates:

$$aep: ({}^V A \xrightarrow{t} Bool) \xrightarrow{t} (A_\varepsilon{}^V \xrightarrow{t} Bool) \text{ and}$$

$$dep: (A_\varepsilon{}^V \xrightarrow{t} Bool) \xrightarrow{t} ({}^V A \xrightarrow{t} Bool).$$

For every $p \in (A_\varepsilon^{\,V} \xrightarrow{\ t\ } Bool)$, $dep(p)$ is defined as follows: given any $d \in {}^{V}A$ $dep(p)(d)=p(ade(d))$. For every $p \in ({}^{V}A \xrightarrow{\ t\ } Bool)$, $aep(p)$ is defined as follows: given any $d \in A_\varepsilon^{\,V}$ $aep(p)(d)=p(dee(d))$.

From definitions we see that

$$dep\colon (A_\varepsilon^{\,V} \xrightarrow{\ t\ } Bool) \xrightarrow{\ t\ } ({}^{V}A \xrightarrow{\ t\ } Bool)$$

is an isomorphism between algebras

$$< (A_\varepsilon^{\,V} \xrightarrow{\ t\ } Bool),\ \vee, \neg,\ R_{\bar{x}}^{\bar{v}} > \text{ and } < ({}^{V}A \xrightarrow{\ t\ } Bool),\ \vee, \neg,\ R_{\bar{x}}^{\bar{v}} >.$$

**Theorem 3.2.**

Let $\Phi \in Fr_{RCNL}(V, Ps)$. Then $|\approx_{RCNL} \Phi$ iff $|\approx_{FCL} clf[urnf[\Phi]]$.

*Proof.*

Let $\Phi_{CL} = clf[urnf[\Phi]]$, $J = ({}^{V}A, TPr({}^{V}A), I)$ be a $\pi$-interpretation, $\Phi_J(d_0)\downarrow = T$ for some $d_0 \in {}^{V}A$. We define $J$ to interpret formulas as total predicates. This was justified earlier.

Now, given $J$, $\Phi$, and $d_0$, we construct a classical interpretation $J_{CL} = (A_\varepsilon^{\,V}, TPr(A_\varepsilon^{V}), I_{CL})$. Interpretation $I_{CL}$ for predicate symbols is defined as follows. Let $(v_1, \ldots, v_n)$ be a list of upper names occurring in $urnf[\Phi]$. Then for every $P \in Ps$ we construct an $n$-ary predicate $p = I_{CL}(P)$ in the following way:

$$p(a_1,\ldots,a_n) =$$
$$I(P)([v \mapsto a \mid v \mapsto a \in_n d_0,\ v \notin \{v_1,\ldots,v_n\}]\nabla[v_i \mapsto a_i \mid i = 1,\ldots,n,\ a_i \neq \varepsilon]).$$

The interpretation $J_{CL}$ induces interpretations of all sub-formulas of $\Phi_{CL}$ into predicate class $(A_\varepsilon^{\,V} \xrightarrow{\ t\ } Bool)$.

Let $D_\varepsilon = \{ade(d_0)\nabla d \mid d \in A_\varepsilon^{V_\Phi}\}$ and $D = \{dee(d) \mid d \in D_\varepsilon\}$. It is clear that *ade* as well as *dee* are bijections between $D_\varepsilon$ and $D$.

We also have that *clf* is a bijection between sub-formulas of $\Phi$ and sub-formulas of $\Phi_{CL}$. Now we prove by induction that any sub-formula $\Psi$ of $\Phi$ and its counterpart $\Psi_{CL} = clf[\Psi]$ are interpreted as predicates that yield the same values on related (by *ade*) elements of $D$ and $D_\varepsilon$ respectively. That is: $\Psi_J(d) = (\Psi_{CL})_{J_{CL}}(ade(d))$, $d \in D$.

Base of induction: Let $\Psi = R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n} P$, then $\Psi_{CL} = P(x_1,\ldots,x_n)$. By construction of $I_{CL}$ we have that

$(R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n} P)_J(d) = (P(x_1,\ldots,x_n))_{J_{CL}}(ade(d))$, because $v_i \in V_\Phi$ $(i = 1,\ldots,n)$. Inductive step is trivial.

Now from $\Phi_J(d_0)\downarrow = T$ we obtain that $(\Phi_{CL})_{J_{CL}}(ade(d_0))\downarrow = T$.

Let us prove the inverse. Suppose $(\Phi_{CL})_{J_{CL}}(d_\varepsilon) = T$ for some $d_\varepsilon \in A_\varepsilon^{\,V}$.

Let us construct an interpretation $J=({}^V A, TPr({}^V A), I)$ and data $d_0$ such that $\Phi(d_0)_J = T$.

Let $D = \{dee(d) \mid d \in A_\varepsilon^{\,V_\Phi}\}$. For each $P$ occurring in $\Phi$ within renominative atoms with $(v_1, \ldots, v_n)$ as upper names and for every $d \in {}^V A$ we assign $I(P)(d) = I_{CL}(P)(ade(d)(v_1),\ldots,ade(d)(v_n))$. Now we can prove in the same way that $\Phi(dee(d_\varepsilon))_J = T$.

Theorem 3.2 allows us to solve the satisfiability problem in RCNL by applying the methods for solving the same problem in classical quantifier-free predicate logic. Obtained formulas of the latter logic can further be transformed into equisatifiable formulas of pure propositional logic. However, this requires the data set $A$ to have enough values to distinguish all $(x_1,\ldots, x_n)$ tuples occurring in $\Phi_{CL}$. Anyhow it is not important when we consider general satisfiability problem, i.e., when we do not concretize the data set. Thus, at the renominative level we can reduce the satisfiability problem in CNL to the propositional satisfiability problem. The reduction is illustrated on the following examples.

**Example 3.2.**

Consider an RCNL formula $\Phi$ with three predicate symbols: $P$, $Q$, $S$. Let the set $V$ contain $x_1$, $y_1$, $x_2$, $y_2$. Let

$$\Phi = R^{x_1}_{y_1}(P \vee Q) \,\&\, \neg R^{x_1,x_2}_{y_1,y_2}(P \vee R^{x_2,y_1}_{x_1,y_2} Q) \,\&\, S\,.$$

According to the procedure, we construct a renominative normal form of Φ by applying simple transformations. Thus, we obtain $\Phi_1$.

$$\Phi_1 = (R^{x_1}_{y_1} P \vee R^{x_1}_{y_1} Q) \,\&\, \neg(R^{x_1,x_2}_{y_1,y_2} P \vee R^{x_1,x_2,y_1}_{y_1,y_1,y_2} Q) \,\&\, R\, S\,.$$

Now we construct URNF by unifying occurrences of the renominative atoms $R^{x_1}_{y_1} P$, $R^{x_1,x_2}_{y_1,y_2} P$, $R^{x_1}_{y_1} Q$, $R^{x_1,x_2,y_1}_{y_1,y_1,y_2} Q$, and $R\, S$. Thus, we obtain $\Phi_2$.

$$\Phi_2 = (R^{x_1,x_2,y_1}_{y_1,x2,y_1} P \vee R^{x_1,x_2,y_1}_{y_1,x_2,y_1} Q) \,\&\, \neg(R^{x_1,x_2,y_1}_{y_1,y_2,y_1} P \vee \; R^{x_1,x_2,y_1}_{y_1,y_1,y_2} Q) \,\&\, R^{x_1,x_2,y_1}_{x_1,x_2,y_1} S$$

Now we are ready to construct a classical predicate logic formula $\Phi_{CL}$ which has ternary predicates *P*, *Q*, *S*.

$$\Phi_{CL} = (P(y_1, x_2, y_1) \vee Q(y_1, x_2, y_1)) \,\&\, \neg(P(y_1, y_2, y_1) \vee Q(y_1, y_1, y_2)) \& S(x_1, x_2, y_1).$$

This formula is satisfiable. For example, we can consider the set of natural numbers as a set *A*. Then the latter formula is equisatisfiable to the following formula of propositional logic

$$(P_1 \vee Q_1) \,\&\, \neg(\, P_2 \vee Q_2) \& S,$$

which is clearly satisfiable. Thus, we have established $|\approx \Phi$. Note that Φ is not satisfiable for data set $\{\alpha\}^V$. Indeed, if we use one-element set then $\Phi_{CL}$ would be equisatisfiable with the propositional formula $(P_1 \vee Q_1) \,\&\, \neg(P_1 \vee Q_1) \,\&\, S$, which is not satisfiable.

**Example 3.3.**

Consider an RCNL formula Φ with two predicate symbols: *P, Q*. Let the set *V* contain $x_1$, $y_1$, $x_2$, $y_2$. Let

$$\Phi = R^{x_1,x_2}_{y_1,y_2}(P \,\&\, Q) \,\&\, \neg R^{x_2}_{y_2}(R^{x_1}_{y_1} P)\,.$$

RNF of Φ is then

$$\Phi_1 = R^{x_1,x_2}_{y_1,y_2} P \,\&\, R^{x_1,x_2}_{y_1,y_2} Q \,\&\, \neg R^{x_1,x_2}_{y_1 y_2} P\,.$$

No unification of renominative atoms needed, so an equisatisfiable predicate logic formula $\Phi_{CL}$ has two binary predicates $p$ and $q$ and is as follows:

$$\Phi_{CL} = p(y_1, y_2) \,\&\, q(y_1, y_2) \,\&\, \neg p(y_1, y_2).$$

This formula is unsatisfiable in FCL, which means that $\Phi$ is unsatisfiable in RCNL.

**Reduction of satisfiability problem for the first-order CNL**

Now we present a result that the satisfiability problem in the first-order CNL can be reduced to the satisfiability problem in the classical first-order predicate logic. This result can be seen as a generalization of the reduction of the satisfiability problem for CNL of quantifier-equational level [Janssen1997].

A formula $\Phi$ is called *satisfiable in an interpretation J* if there is $d \in {}^{V}A$ such that $\Phi_J(d)\downarrow = T$. We shall denote this by $J \models \Phi$. A formula $\Phi$ is called *satisfiable* if there exists an interpretation $J$ in which $\Phi$ is satisfiable. We shall denote this as $\models \Phi$. We call formulas $\Phi$ and $\Psi$ *equisatisfiable* if they are either both satisfiable or both not satisfiable (i.e., unsatisfiable).

The problem is to check whether $\models \Phi$ holds given an arbitrary formula $\Phi \in Fr(V, Fs, Ps)$. Our main aim is to transform this CNL formula $\Phi$ to an equisatisfiable formula $\Phi_{CL}$ of the classical first-order logic with equality so that we can use existent methods for solving this problem developed for classical logics.

We transform the formula $\Phi$ to the formula $usnf[\Phi]$ in a special normal form, which is translated to its classical counterpart $\Phi_{CL}$ afterwards.

A formula $\Phi$ is said to be in *unified superposition normal form* (USNF) if the following requirements are met:

– for every sub-formula of the form $S_P^{\bar{v}}(\Psi, \bar{t}\,)$ we have that $\Psi \in Ps$;

– for every sub-formula of the form $S_F^{\bar{v}}(r, \bar{t}\,)$ we have that $r \in Fs$;

– all instances of superposition compositions have the same list of variables $\bar{w}$;

– for every quantifier $\exists y$ occurring in $\Phi$ we have that $y$ should occur in $\bar{w}$ (see the previous rule).

Consider **equivalent** transformation rules T1-T12 of the form $\Phi_l \mapsto \Phi_r$, $\Phi_l, \Phi_r \in Fr(V, Fs, Ps)$. These rules are equivalent transformations in CNL.

T1) $S_P^{\bar{v}}(\Phi \vee \Psi, \bar{t}\,) \mapsto S_P^{\bar{v}}(\Phi, \bar{t}\,) \vee S_P^{\bar{v}}(\Psi, \bar{t}\,)$.

T2) $S_P^{\bar{v}}(\neg\Phi, \bar{t}\,) \mapsto \neg S_P^{\bar{v}}(\Phi, \bar{t}\,)$.

T3) $S_P^{\bar{v}}(\exists x\Phi,\ \bar{t}\,) \mapsto \exists u\ S_P^{\bar{v}}(S_P^{x}(\Phi, 'u),\ \bar{t}\,)$, $u$ is an unessential variable that does not occur in $S_P^{\bar{v}}(\exists x\Phi,\ \bar{t}\,)$, $u \in U$.

T4) $S_P^{\bar{u},\bar{x}}(S_P^{\bar{x},\bar{v}}(\Phi,\ \bar{r}, \bar{s}\,),\ \bar{t}, \bar{w}\,) \mapsto S_P^{\bar{u},\bar{x},\bar{v}}(\Phi,\ \bar{t}\,,\ S_F^{\bar{u},\bar{x}}(r_1,\ \bar{t}\,,\ \bar{w}\,), \ldots,$

$S_F^{\bar{u},\bar{x}}(r_k,\ \bar{t}\,,\ \bar{w}\,),\ S_F^{\bar{u},\bar{x}}(s_1,\ \bar{t}\,,\ \bar{w}\,), \ldots,\ S_F^{\bar{u},\bar{x}}(s_m,\ \bar{t}\,,\ \bar{w}\,))$, here (and in T5)

$\bar{u} = u_1,\ldots,u_n;\ \ \bar{t} = t_1,\ldots,t_n;\ \ \bar{x} = x_1\ ,\ldots,\ x_k;\ \ \bar{r} = r_1,\ldots,r_k;\ \ \bar{w} = w_1,\ldots,w_k;\ \ \bar{v} = v_1,\ldots,v_m;\ \ \bar{s} = s_1,\ldots,s_m,\ u_i \neq v_j, i = 1,\ldots,n, j = 1,\ldots,m$.

T5) $S_F^{\bar{u},\bar{x}}(S_F^{\bar{x},\bar{v}}(t,\ \bar{r}, \bar{s}\,),\ \bar{t}, \bar{w}\,) \mapsto S_F^{\bar{u},\bar{x},\bar{v}}(t,\ \bar{t}\,,\ S_F^{\bar{u},\bar{x}}(r_1,\ \bar{t}\,,\ \bar{w}\,), \ldots,\ S_F^{\bar{u},\bar{x}}(r_k,\ \bar{t}\,,\ \bar{w}\,),$

$S_F^{\bar{u},\bar{x}}(s_1,\ \bar{t}\,,\ \bar{w}\,), \ldots,\ S_F^{\bar{u},\bar{x}}(s_m,\ \bar{t}\,,\ \bar{w}\,))$.

T6) $S_P^{\bar{v}}(r = s,\ \bar{t}\,) \mapsto S_F^{\bar{v}}(r,\ \bar{t}\,) = S_F^{\bar{v}}(s,\ \bar{t}\,)$.

T7) $S_P^{\bar{v}}(\Phi,\ \bar{t}\ ) \mapsto S_P^{x,\bar{v}}\ (\Phi,'x,\bar{t}\ )$, $x$ does not occur in $\bar{v}$. In particular $\Phi \mapsto S_P^x(\Phi,\ 'x)$.

T8) $S_F^{\bar{v}}(t,\ \bar{t}\ ) \mapsto S_F^{x,\bar{v}}(t,\ 'x,\ \bar{t}\ )$, $x$ does not occur in $\bar{v}$. In particular, $t \mapsto S_F^x\ (t,\ 'x)$.

T9) $S_P^{\bar{u},x,\bar{v}}(\Phi,\bar{q},r,\bar{s}\ ) \mapsto S_P^{x,\bar{u},\bar{v}}(\Phi,\ r,\bar{q},\ \bar{s}\ )$.

T10) $S_F^{\bar{u},x,\bar{v}}(t,\bar{q},r,\bar{s}\ ) \mapsto S_F^{x,\bar{u},\bar{v}}(t,\ r,\bar{q},\ \bar{s}\ )$.

T11) $S_F^{x,\bar{v}}('x,t,\bar{r}) \mapsto t$.

T12) $S_F^{\bar{v}}('x,\bar{r}) \mapsto 'x$, $x$ does not occur in $\bar{v}$.

Rules T3 and T7 permit to assume without loss of generality that all quantified variables in initial formula are different.

Given an arbitrary formula $\Phi \in Fr(V,Fs,Ps)$ we can construct (non-deterministically) its unified superposition normal form $usnf[\Phi]$ by applying rules T1–T12. This transformation is satisfiability-preserving.

In order to reduce the satisfiability problem in the first-order CNL to the satisfiability problem of classical logics we consider algebras defined over extended data set $A_\varepsilon^V = V \xrightarrow{t} A \cup \{\varepsilon\}$. Informally, the additional value $\varepsilon$ represents undefined components of nominative data. So, by changing data classes from ${}^VA$ to $A_\varepsilon^V$ we make our algebras closer to their classical counterparts and simplify required proofs.

We formalize the syntactical reduction *clf* of terms and formulas in unified superposition normal form to formulas of classical logic inductively as follows:

1. $clf['x] \mapsto x$.
2. $clf[S_F^{w_1,\ldots,w_n}(F,t_1,\ldots,t_n)] \mapsto F(clf[t_1],\ldots,clf[t_n])$, $F \in Fs$.
3. $clf[(\Phi_1 \vee \Phi_2)] \mapsto (clf[\Phi_1] \vee clf[\Phi_2])$.
4. $clf[\neg\Phi] \mapsto \neg clf[\Phi]$.

5. $clf[S_P^{w_1,\ldots,w_n}(P,t_1,\ldots,t_n)] \mapsto P(clf(t_1),\ldots,clf(t_n))$, $n \geq 0$.

6. $clf[\exists x\Phi] \mapsto \exists x(x \neq e \,\&\, clf[\Phi])$, $e \in U$, $e$ is a predefined variable.

7. $clf[t_1 = t_2] \mapsto clf(t_1) = clf(t_2)$.

Note that all applications of the 6-th rule introduce the same variable $e$ which is a predefined variable from $U$ in the sense that it does not occur in USNF. In interpretations the value of this variable is treated as $\varepsilon$.

This reduction transforms the formula to the language of classical logic but preserves its satisfiability.

**Theorem 3.3.** Let $\Phi \in Fr(V, Fs, Ps)$. Then $|\approx \Phi$ if and only if $clf[usnf[\Phi]]$ is satisfiable in the classical first-order predicate logic.

The theorem states the reduction of the satisfiability problem in the first-order composition-nominative logic to the satisfiability problem in the classical first-order predicate logic.

### *3.4. First-order logic of partial quasiary predicates with the complement composition*

We treat a logic $L$ as a tuple $(\mathcal{A}, Fr, \mathcal{I}, |=, |–)$ [NikSh2013] where

– $\mathcal{A}$ is *a class of algebras of some signature*;

– $Fr$ is *a language (based on the algebra signature)*;

– $\mathcal{I}$ is *a class of interpretations*;

– $|=$ is *a consequence relation*;

– $|–$ is *an inference relation based on some calculus*.

Here we consider only pure (without functions) logic $L^{QEC}$. This logic is the next step of our construction of series of first-order logics of partial quasiary predicates.

Earlier, we started with a basic logic $L^Q$ with compositions of *disjunction* $\vee$, *negation* $\neg$, *renomination* $R_{\bar{x}}^{\bar{y}}$, and *existential quantifica-*

*tion* $\exists x$. This logic was not expressive enough to prove its completeness therefore a logic $L^{QE}$ was constructed as an extension of $L^{Q}$ with *the null-ary parametric composition* (*predicate*) *Ez* of *variable* assignment. (Also, *variable unassignment predicate* $\varepsilon z$ can be used [NikTym2012].) But again, logic $L^{QE}$ was not expressive enough to construct sound program logics of Floyd-Hoare type therefore $L^{QE}$ is extended to a new logic $L^{QEC}$ by adding *the composition of predicate complement* $\sim$ (discussion on the topic is presented in [Abramsky1993]).

**Predicate Algebras with the Complement Composition**

Let $V$ and $A$ be sets of *names* (*variables*) and *values* respectively. The class ${}^{V}A$ of *nominative sets* (*partial assignments, partial data*) is defined as the class of all partial mappings from $V$ to $A$, thus, ${}^{V}A = V \xrightarrow{p} A$.

The main operation for nominative sets is a total unary parametric *renomination* $\mathrm{r}^{v_1,...,v_n}_{x_1,...,x_n} : {}^{V}A \xrightarrow{t} {}^{V}A$, where $v_1,...,v_n, x_1,...,x_n$ are names, and $v_1,...,v_n$ are distinct [NikSh2013]. Intuitively, given a nominative set $d$ this operation yields a new nominative set changing the values of $v_1,...,v_n$ to the values of $x_1,...,x_n$ respectively. For this operation we also use simpler notation $\mathrm{r}^{\bar{v}}_{\bar{x}}$; $x \in \bar{v}$ means that $x$ is a variable from $\bar{v}$; $\bar{v} \cup \bar{x}$ is the set of variables that occur in $\bar{v}$ and $\bar{x}$; *asn*($d$) is the set of *assigned variables* (*names*) in $d$.

The set $Pr^{V}_{A} = {}^{V}A \xrightarrow{p} Bool$ is called the set of *partial quasiary predicates*. For a predicate $p \in Pr^{V}_{A}$ its *truth, falsity,* and *undefinedness domains* are denoted $T(p)$, $F(p)$, and $\perp(p)$ respectively. Note, that a predicate $p$ is defined by $T(p)$ and $F(p)$ only, because $\perp(p) = {}^{V}A \setminus (T(p) \cup F(p))$.

A name (variable) $z$ is *unessential* for $p \in Pr^{V}_{A}$, if for any $d \in {}^{V}A$ the value of $p$ does not depend on the value of $z$.

Operations over $Pr_A^V$ are called *compositions*. Basic compositions of first-order level of quasiary predicates are *disjunction* $\vee$, *negation* $\neg$, *renomination* $R_{\bar{x}}^{\bar{y}}$, and *existential quantification* $\exists x$.

We define them via their definedness domains ( $p, q \in Pr_A^V$ ):

– $T(p \vee q) = T(p) \cup T(q)$, $F(p \vee q) = F(p) \cap F(q)$;

– $T(\neg p) = F(p)$, $F(\neg p) = T(p)$;

– $T(R_{\bar{x}}^{\bar{y}}(p)) = \{d \in {}^V A \mid \mathrm{r}_{\bar{x}}^{\bar{y}}(d) \in T(p)\}$,

$F(R_{\bar{x}}^{\bar{y}}(p)) = \{d \in {}^V A \mid \mathrm{r}_{\bar{x}}^{\bar{y}}(d) \in F(p)\}$;

– $T(\exists x\, p) = \{d \in {}^V A \mid d \nabla x \mapsto a \in T(p) \text{ for some } a \in A\}$,

$F(\exists x\, p) = \{d \in {}^V A \mid d \nabla x \mapsto a \in F(p) \text{ for all } a \in A\}$.

Please note that definitions of disjunction and negation are similar to *strong Kleene's connectives*; their properties are described in [Kleene1952].

Also we use *variable* assignment predicate *Ez* defined as follows:

$$T(Ez) = \{d \mid d(z) \downarrow\} = \{d \in {}^V A \mid z \in asn(d)\},$$
$$F(Ez) = \{d \mid d(z) \uparrow\} = \{d \in {}^V A \mid z \notin asn(d)\}.$$

At last, the complement composition is defined in the following way:

$$T(\sim p) = \bot(p), \quad F(\sim p) = \varnothing.$$

We consider $\sim$ as a composition of propositional level. This composition differs from traditional compositions. The main difference lies in the fact that traditional compositions are applicative composition [Nik2001]. *Applicativity* of composition *C* means that given predicates $p_1, \ldots, p_n$ the value of $C(p_1, \ldots, p_n)$ on some data is evaluated upon values of $p_1, \ldots, p_n$ on data from their definedness domains. The complement composition is not applicative because the value of $\sim p$ on some *d* may depend upon undefinedness domain of *p*. This fact complicates logics with such composition because the undefinedness domains should be explicitly involved in the definitions of consequence relations.

Note, that applicative compositions are monotone with respect to predicate graph inclusion; but composition $\sim$ is not.

A tuple $\mathcal{A}^{QEC}(V, A) = < Pr_A^V ; \vee, \neg, R_{\bar{x}}^{\bar{v}}, \exists x, Ez, \sim >$ is called *a first-order complemented algebra of partial quasiary predicates.*

A class of such algebras (with different *A*) forms a semantic base for logic $L^{QEC}$.

Now we describe the main properties of $\mathcal{A}^{\mathrm{QEC}}$(V, A). We concentrate on the complement composition and do not formulate properties of disjunction, negation, renomination, existential quantification, and unassignment predicate [NikSh2017, NikSh2019].

**Lemma 3.10.** *For any* $p \in Pr_A^V$ *we have*

– $\sim p$ ; $\sim\sim\sim p = \sim p$ ; $\sim\sim\sim\sim p = \sim\sim p$ ;

– $T(\neg\sim p) = \varnothing$ ; $F(\neg\sim p) = T(\sim p)$ ; $\perp(\neg\sim p) = \perp(\sim p)$ ;

– $T(R_{\bar{x}}^{\bar{v}}(\sim p)) = T(\sim(R_{\bar{x}}^{\bar{v}}(p)))$ ; $F(R_{\bar{x}}^{\bar{v}}(\sim p)) = \varnothing$ ;

$\perp(R_{\bar{x}}^{\bar{v}}(\sim p)) = \perp(\sim(R_{\bar{x}}^{\bar{v}}(p)))$ ;

– $T(\sim\exists xP) = \perp(\exists xP)$ ; $F(\sim\exists xP) = \varnothing$ ;

$\perp(\sim\exists xP) = T(\exists xP) \cup F(\exists xP)$ ;

– $R_{\bar{x},y}^{\bar{v},z}(Ez) = Ey$; $R_{\bar{x}}^{\bar{v}}(Ez) = Ez$, if $z \notin \{\bar{v}\}$.

**Language (signature and formulas) of $\mathbf{L^{QEC}}$**

*Let V be an infinite set of* names *(*variables*) and U be an infinite subset of V called* a set of unessential variables. *Let* Ps *be a set of* predicate symbols. *A tuple* $\Sigma^{QEC} = (V, U; \vee, \neg, R_{\bar{x}}^{\bar{v}}, \exists x, Ez, \sim; Ps)$ is called the *language signature*.

For simplicity, we use the same notation for symbols of compositions and compositions themselves.

Given $\Sigma^{QEC}$, we define inductively the *language* of $L^{QEC}$ – the *set of formulas* denoted $Fr(L^{QEC})$ or simply *Fr*:

– if $P \in Ps$ then $P \in Fr$. Formulas of such forms are called *atomic*;

– $Ez \in Fr$;

– if $\Phi, \Psi \in Fr$ then $\Phi \vee \Psi$, $\neg\Phi$, $R_{\bar{x}}^{\bar{v}}\Phi$, $\exists x\Phi$, $\sim\Phi \in Fr$ .

Formulas of the form $R_{\bar{x}}^{\bar{y}}\Phi$ are called R-formulas.

Further definitions and inference relation for this logic is given in Chapter 6.

### *3.5. Questions and exercises*

1. What is a quasiary function?
2. What are the types of carriers in predicate algebras?
3. What are renominative logics?
4. What are the properties of renomination composition?
5. What are the compositions of first-order logics?
6. How are quantification compositions defined?
7. What are composition-nominative logics?
8. How are composition-nominative logics defined?
9. How are composition-nominative logics of a propositional-level defined?
10. How are composition-nominative logics of a renominative level defined?
11. How are composition-nominative logics of a quantifier level defined?
12. How are propositional classical logics defined?
13. How are classical quantifier-free logics defined?
14. How are first-order classical predicate logics defined?
15. What is the satisfiability problem?
16. How is the satisfiability problem solved for propositional logics?
17. How is the satisfiability problem solved for renominative logics?
18. How is the satisfiability problem solved for predicate logics?
19. What is the predicate complement composition?
20. What are the components of logic?

21. How are predicate algebras with predicate complement composition defined?
22. How is the language of predicate logic with predicate complement composition defined?

## Chapter 4. Hierarchical nominative data

Nominative data can be treated as a universal mathematical model of data [NikIv2010]. They are based on the name-value relation, and allow one to conveniently represent many data structures frequently used in programming. In the simplest case nominative data can be considered as associations of names with values, and in more complex cases they are hierarchical structures.

### *4.1. Different types of hierarchical nominative data*

Nominative data are built over classes of names *V* and atoms *A* with the help of naming relations. Thus in the first approximation, a nominative data *d* is either an atom *A*, or an expression of the form $[v_1 \mapsto d_1,...,v_n \mapsto d_n]$, where $v_1,...,v_n$ are different names from *V*, and $d_1, d_2,..., d_n$ – are atoms or other nominative data.

To define nominative data formally let us denote by $V \xrightarrow{n} B$ the class of all partial functions from *V* to a set of values *B*, which has a finite graph.

We classify nominative data in accordance with the following parameters:

- *values* can be abstract (simple, unstructured) or complex (structured),
- *names* can be abstract (simple, unstructured or complex (structured).

The possible values of parameters give four types of nominative data. Let us clarify the notion of a complex name and a complex value.

Data with *complex values* are nominative data in which values associated with names can themselves be nominative data.

To clarify the notion of a *complex name* we will follow the principle of development (from the abstract to the concrete) and consid-

er an abstract form of construction and processing of names. The main operation on complex names will be concatenation. This operation is associative.

*Principle of associative construction and processing of complex names*: complex names are constructed from abstract names using concatenation, and data with complex names have to be processed using operations that take into account associativity of concatenation of names.

Besides, we will require that data with complex names satisfy the following principle.

*Principle of unambiguous associative naming*: one complex name should have no more than one corresponding value in a given data.

Taking into account the mentioned principles let us give formal (mathematical) definitions of nominative data of different types.

**1. The class of nominative data with abstract names and abstract values** over a nonempty set of names $V$ and a nonempty set of abstract values $A$ is:

$$D_0 = V \xrightarrow{n} A.$$

This class contains data of the most abstract type. For example, if $u, v$ are abstract names from $V$, and $a, b$ are values from $A$, then

$$[u \mapsto a, v \mapsto b] \in D_0.$$

We will call the element of $D_0$ data of the type $TND_{AA}$ (data with abstract names and values).

**2. The class of nominative data with abstract names and complex values** over a nonempty set of names $V$ and a nonempty set of abstract values $A$ is:

$$D_1 = ND(V, A),$$

here $ND(V, A)$ is

$$ND(V, A) = \bigcup_{k \geq 0} ND_k(V, A),$$

where

$$ND_0(V, A) = A \cup \{\varnothing\},$$

$$ND_{k+1}(V, A) = A \cup \left(V \xrightarrow{n} ND_k(V, A)\right), \quad k \geq 0.$$

Here we denote as $\varnothing$ the empty nominative data (the same notation is used for the empty set). For the empty nominative data, we will also use the notation [].

Data of this class have hierarchical structure, e.g. if $u, v, w \in V$ and $a, b \in A$, then

$$[u \mapsto a, v \mapsto [w \mapsto b]] \in ND(V, A).$$

Such data can be represented as oriented trees with arcs labeled by names and leafs labeled by atoms or empty data. A *path* is a nonempty finite sequence $(v_1, v_2, ..., v_k)$ of names $v_1, ..., v_k \in V$. For any data $d$, the *value of the path* $(v_1, v_2, ..., v_k)$ in $d$ is the value of the expression

$$d(v_1, v_2, ..., v_k) \cong (...((d(v_1))(v_2))...(v_k)).$$

We say that a path $(v_1, v_2, ..., v_k)$ is a *path in data* $d \in ND(V, A)$, if the value $(v_1, v_2, ..., v_k)$ in $d$ is defined, i.e. $d(v_1, v_2, ..., v_k)\downarrow$.

A *terminal path* in a data $d \in ND(V, A)$ is a path in $d$ such that its value belongs to $A \cup \{\varnothing\}$.

The least $k$ such that $d \in ND_k(V, A)$ is called the *rank* of $d$.

The elements of $D_1$ are called the data of type $TND_{AC}$ (data with abstract names and complex values).

**3. Class of nominative data with complex names and abstract values** over a nonempty set of names $V$ and a nonempty set of abstract values $A$ is:

$$D_2 = NDVS(V, A),$$

where $NDVS(V, A)$ is a set of all elements $A \cup (V^+ \xrightarrow{n} A)$ such that either $d \in A$, or $d \in V^+ \xrightarrow{n} A$ and all words in $dom(d)$ are pairwise

incomparable in the sense of the prefix relation (*the principle of unambiguous associative naming*).

For example, if $u,v,w \in V$ and $a,b,c \in A$, then

$$[uv \mapsto a, uw \mapsto b, w \mapsto c] \in NDVS(V,A).$$

The data of this class have complex names, i.e. the names are words in the alphabet *V*.

The elements of $D_2$ are called data of type $TND_{CA}$ (data with complex names and abstract values).

**4. The class of nominative data with complex names and complex values** over a nonempty set of names $V$ and a nonempty set of abstract values $A$ is:

$$D_3 = NDVC(V,A),$$

where $NDVC(V,A)$ is the class of all data $d \in ND(V^+,A)$ such that for any two paths $(u_1,u_2,...,u_k)$ and $(v_1,v_2,...,v_l)$ in *d*, neither of which is a prefix of another one, the words $u_1u_2...u_k$ and $v_1v_2...v_l$ are incomparable in the sense of the prefix relation (the principle of unambiguous associative naming). Such data are called *complex-named data*. For example, if $a \in A$ and $u,v,w \in V$, then

$$[uv \mapsto a, w \mapsto [uw \mapsto \varnothing]] \in NDVC(V,A).$$

Such data are hierarchical and have complex names and unambiguous naming.

The elements of $D_3$ are called data of type $TND_{CC}$ (data with complex names and complex values).

### *4.2. Representation of data structures by nominative data*

Let us give some arguments in support of the following principle:

*The principle of representative completeness of nominative data*: different forms of data used in computer information processing systems can be adequately represented as nominative data

The examples are given below. Representations are chosen in such a way that basic operations of the mentioned structures correspond to simple operations on nominative data.

**1. Array** $(a_1, a_2, ..., a_n)$ of elements of type $T_0$.

The type of nominative data is:

$$TND_{AA},\ V = \{1,2,\ldots\},\ A = T_0.$$

Representation:

$$[1 \mapsto a_1, 2 \mapsto a_2, \ldots, n \mapsto a_n].$$

**2. Two-dimensional array** $\begin{pmatrix} a_{1,1} & a_{1,2} & ... & a_{1,m} \\ ... & ... & ... & .... \\ a_{n,1} & a_{n,2} & ... & a_{n,m} \end{pmatrix}$ of elements of type $T_0$.

The type of nominative data is:

$$TND_{CA} \text{ (complex names)},\ V = \{1,2,\ldots\},\ A = T_0.$$

Representation:

$$[i.j \mapsto a_{i,j} \mid i = 1,\ldots,n,\ j = 1,\ldots,m]$$

(here "." denotes the concatenation of characters which forms a complex name).

**3. An array of arrays** $((a_{i,j})_{j=1}^{m})_{i=1}^{n}$ of elements of type $T_0$.

The type of nominative data is:

$$TND_{AC},\ V = \{1,2,\ldots\},\ A = T_0.$$

Representation:

$$[i \mapsto [\, j \mapsto a_{i,j} \mid j = 1,\ldots,m] \mid i = 1,\ldots,n].$$

**4. An associative array** $(a_k)_{k \in K}$ of elements of type $T_0$, where $K$ is a set of keys.

The type of nominative data is:

$$TND_{AA},\ V = K,\ A = T_0.$$

Representation:

$$[k \mapsto a_k \,|\, k \in K].$$

**5. A table**, where **Key** is a key attribute:

| **Key** | **Attr1** | **Attr2** | ... | **AttrN** |
|---|---|---|---|---|
| $key_1$ | $val_{11}$ | $val_{21}$ | ... | $val_{N1}$ |
| $key_2$ | $val_{12}$ | $val_{22}$ | ... | $val_{N2}$ |
| ... | ... | ... | ... | ... |
| $key_M$ | $val_{1M}$ | $val_{2M}$ | ... | $val_{NM}$ |

The type of nominative data is:

$$TND_{AC}\,,\ V = \{Attr_1, Attr_2, ..., Attr_M\}\,.$$

Representation:

$$[key_1 \mapsto [Attr_1 \mapsto val_{11}, Attr_2 \mapsto val_{12}, ...Attr_M \mapsto val_{1M}],$$
$$key_2 \mapsto [Attr_1 \mapsto val_{21}, Attr_2 \mapsto val_{22}, ...Attr_M \mapsto val_{2M}]]\,.$$
$$.....$$
$$key_N \mapsto [Attr_1 \mapsto val_{N1}, Attr_2 \mapsto val_{N2}, ...Attr_M \mapsto val_{NM}]].$$

**6. A linked list** of elements $e_1, e_2, \ldots e_n$.

The type of nominative data is: $TND_{AC}\,,\ V = \{data, next\}$.

Representation:

$$[data \mapsto e_1, next \mapsto [data \mapsto e_2, next \mapsto$$
$$\mapsto [\ldots data \mapsto e_n, next \mapsto \varnothing \ldots]]].$$

**7. A cyclic list** of elements $e_1, e_2, \ldots e_n$.

The type of nominative data is: $TND_{AC}$,

$$V = \{head, list, data, next\} \cup \{x_1, x_2, x_3, \ldots x_n\}\,.$$

Here $x_1$, $x_2$, ... are auxiliary names which represent the location of the elements of the list.

Representation:

$$[head \mapsto loc_1, list \mapsto [$$
$$x_1 \mapsto [data \mapsto e_1, next \mapsto x_2],$$
$$x_2 \mapsto [data \mapsto e_2, next \mapsto x_3],$$
$$...$$
$$x_n \mapsto [data \mapsto e_n, next \mapsto x_1]]].$$

**8. A binary tree** with nodes labeled by the elements $e_1, e_2, e_3, ...$

The type of nominative data is: $TND_{AC}$, $V = \{data, left, right\}$.

Representation:

$$[data \mapsto e_1,$$
$$left \mapsto [data \mapsto e_2, left \mapsto [...], right \mapsto [...]],$$
$$right \mapsto [data \mapsto e_3, left \mapsto [...], right \mapsto [...]$$
$$]].$$

**9. Inductive data types.** Data of such types are widely used in functional programming languages. They can be interpreted as terms which satisfy special conditions.

Let $S$ be a finite set of sorts. Let $\Sigma$ be a finite signature $\{c_1, c_2, ..., c_n\}$, where $c_i$ are *constructor* names. Assume that each constructor $c_i$ has a (unique) associated *type* of the form $s_1 \times ... \times s_m \to s$, where $s_i, s \in S$, $m \geq 0$. Let $D$ be a free algebra of the signature $\Sigma$ freely generated by some set $A$. Then the elements of the carrier $D$ can be considered as elements of an inductive data type. Such data can be constructed by applying constructors.

Let us define a mapping $\tau$ which gives a representation of a data from $D$ in the form of a nominative data of type $TND_{AC}$ over the class of names $V = \{constructor, 1, 2, 3, ...\}$ and atoms $A$:

- $\tau(x) = x$, if $x \in A$;
- $\tau(x) = [constructor \mapsto c, 1 \mapsto \tau(x_1), 2 \mapsto \tau(x_2), ..., n \mapsto \tau(x_n)]$, if $x$ has the form $c(x_1, x_2, \ldots, x_n)$, where $c \in \Sigma$.

The application of constructors to such data corresponds to the naming operation on nominative data.

### *4.3. Algebras of nominative data*

The main operations on nominative data include:

- *denomination* (obtains a value which corresponds to a name, also called denaming operation),
- *nomination* (gives a new value to a name),
- *overlapping* (joins two data with priority of the second data).

Let us define these operations formally for data of types $TND_{CA}$, $TND_{AC}$, and $TND_{CC}$. Similar operations can be easily defined for data of type $TND_{AA}$.

We will use the same notation for denomination, nomination, and overlapping for each type $TND_{CA}$, $TND_{AC}$, and $TND_{CC}$. The interpretation of symbols of operations should be clear from the context.

Let $V$ and $A$ be fixed sets of names and atoms.

**Definition 4.1 (denomimation [**SkobNikIv2014**]).**

1. For nominative data of type $TND_{AC}$ *denomination* is a unary operation $v \Rightarrow_a$ with parameter $v \in V^+$ defined by induction on the length of $v$ (the length of a word is denoted as $|v|$) as follows:

- if $|v| = 1$, then $v \Rightarrow_a (d) \cong d(v)$;
- if $|v| = n > 1$, then $v \Rightarrow_a (d) \cong v_1 \Rightarrow_a (x \Rightarrow_a (d))$, where $v = xv_1$, $x \in V$, $v_1 \in V^{n-1}$.

For nominative data of types $TND_{CA}$ and $TND_{CC}$ *associative denomination* is a unary operation $v \Rightarrow_a$ with parameter $v \in V^+$ de-

fined by induction on the length of $v$ as follows:

- if $v \in V$, then

$$v \Rightarrow_a (d) \cong \begin{cases} d(v), & d(v)\downarrow; \\ d/v, & d(v)\uparrow \text{ or } d/v \neq \varnothing; \\ \text{undefined}, & d(v)\uparrow \text{ or } d/v = \varnothing, \end{cases}$$

where $d/u = [v_1 \mapsto d(v) \mid d(v)\downarrow, v = uv_1, v_1 \in V^+]$ (a *division* of a data by name);

- if $v \in V^n$ and $n > 1$, then $v \Rightarrow_a (d) \cong v_1 \Rightarrow_a (x \Rightarrow_a (d))$, where $v = xv_1$, $x \in V$, $v_1 \in V^{n-1}$.

The following example illustrates these operations:

$$u \Rightarrow_a ([u \mapsto 1, v \mapsto 2]) = 1;$$
$$(uv) \Rightarrow_a ([u \mapsto [vw \mapsto 1, u \mapsto 2]]) = [w \mapsto 1].$$

The name of the operation (associative denomination) is related to the following property, which is called *associativity of denomination*:

$$u \Rightarrow_a (d) \cong u_n \Rightarrow_a (u_{n-1} \Rightarrow_a (...u_1 \Rightarrow_a (d)...))$$

for all $u, u_1, u_2, ..., u_n \in V^+$ such that $u = u_1 u_2 ... u_n$.

**Definition 4.2 (nomination).**

1. For nominative data of type $TND_{AC}$ *nomination* is a unary operation $\Rightarrow v$ with a parameter $v \in V^+$, defined by induction on the length of $v$ as follows:

- $\Rightarrow v(d) = [v \mapsto d]$, if $v \in V$;
- $\Rightarrow v(d) = [v_1 \mapsto (\Rightarrow v_2(d))]$, if $v = v_1 v_2$, $v_1 \in V$ and $v_2 \in V^+$.

2. For nominative data of type $TND_{CA}$ *nomination* is a unary operation $\Rightarrow v$ with a parameter $v \in V^+$, defined by induction

as follows:

- $\Rightarrow v(d) = [v \mapsto d]$, if $d \in A \cup \{\varnothing\}$;
- $\Rightarrow v(d) = [vu \mapsto d(u) \mid u \in dom(d)]$, if $d \notin A \cup \{\varnothing\}$.

3. For nominative data of type $TND_{CC}$, *nomination* is a unary operation $\Rightarrow v$ with a parameter $v \in V^+$ such that

$$\Rightarrow v(d) = [v \mapsto d].$$

*Overlapping* is an operation which updates the value in its first argument with the values defined in the second argument which correspond to the respective names.

For nominative data with complex names and/or values, one can consider different kinds of overlapping. We will consider two kinds of overlapping — *global* and *local* overlapping.

The global (associative or structural) overlapping $\nabla_a$ updates several values in the first argument, while the local overlapping $\nabla_a^v$ (with name parameter $v$) updates only one value which corresponds to the name $v$. The global overlapping can be used to formalize the semantics of procedure call, and local overlapping can be used to formalize the semantics of the assignment operator in programming languages. Informally, this operation joins two data and resolves name conflict in favor of the *second* argument.

**Definition 4.3 (global overlapping ).**

1. For nominative data of type $TND_{AC}$ *global overlapping* is a partial binary operation $\nabla_a$ such that

- $d_1 \nabla_a d_2 = d_2 \cup d_1 \mid_{dom(d_1) \setminus dom(d_2)}$, if $d_1 \notin A$ and $d_2 \notin A$;
- $d_1 \nabla_a d_2 \uparrow$, if $d_1 \in A$ or $d_2 \in A$.

2. For nominative data of type $TND_{CA}$ *global overlapping* is a binary operation $\nabla_a$ such that

- $d_1 \nabla_a d_2 = d_2 \cup d_1 \mid_{dom(d_1) \setminus (dom(d_2) V^*)}$;
- $d_1 \nabla_a d_2 \uparrow$, if $d_1 \in A$ or $d_2 \in A$,

where $dom(d_2)V^*$ denotes the set of all words of the form $uv$, where $u \in dom(d_2)$ and $v \in V^*$.

3. For nominative data of type $TND_{CC}$, *global overlapping* is a partial binary operation $\nabla_a$, defined inductively by the rank of the first argument in the following way. Let

$$NDVC_k(V,A) = NDVC(V,A) \cap ND_k(V^+,A)$$

be the set of data from $NDVC(V,A)$ the rank of which does not exceed $k$.

*Induction base.* If $d_1 \in NDVC_0(V,A)$, then

$$v \Rightarrow_a (d) \cong \begin{cases} d_2, & d_1 = \varnothing \wedge d_2 \in NDVC(V,A) \setminus A; \\ \text{undefined}, & d_1 \in A \vee d_2 \in A. \end{cases}$$

*Induction step.* Assume that the value $d_1 \nabla_a d_2$ is already defined for all $d_1, d_2$ such that $d_1 \in NDVC_k(V,A)$. Let

$$d_1 \in NDVC_{k+1}(V,A) \setminus NDVC_k(V,A).$$

Then $d_1 \nabla_a d_2 = d$, where the data $d$ is defined on names $u \in V^+$ as follows:

1) $d(u) = d_2(u)$, if $u \in dom(d_2)$ and $u$ does not have a proper prefix which belongs to $dom(d_1)$;

2) $d(u) = d_1(u) \nabla_a (d_2 / u)$, if $d_1(u)$ is defined and does not belong to $A$, and $u$ is a proper prefix of some element of $dom(d_2)$, where $d_2 / u = [v_1 \mapsto d_2(v) \mid d_2(v)\downarrow, v = uv_1, v_1 \in V^+]$ is division of data by name;

3) $d(u) = d_2 / u$, if $d_1(u)$ is defined and belongs $A$, and $u$ is a proper prefix of some element of $dom(d_2)$;

4) $d(u) = d_1(u)$, if $d_1(u)$ is defined and the word $u$ is incomparable (in the sense of prefix relation) with each element of $dom(d_2)$;

5) $d(u)\uparrow$ otherwise.

The global overlapping on data of type $TND_{CC}$ has the following properties [NikIv2010a]:

- $[u \mapsto d_1]\nabla_a[v \mapsto d_2] = [u \mapsto d_1, v \mapsto d_2]$, $u, v \in V$, $u \neq v$;
- $[uv \mapsto d_1]\nabla_a[u \mapsto d_2] = [u \mapsto d_2]$, $u, v \in V^+$, i.e. the value which corresponds to the name $u$ in the 2-nd argument overrides the values of names in the 1-st argument which are extensions of $u$;
- $[u \mapsto d_1]\nabla_a[uv \mapsto d_2] = [u \mapsto (d_1\nabla_a[v \mapsto d_2])]$, if $u, v \in V^+$, $d_1 \notin A$, i.e. the value which corresponds to the name $uv$ in the second argument modifies the value of the names-prefixes of $uv$ in the first argument;
- $\varnothing\nabla_a d = d\nabla_a\varnothing = d$, if $d \notin A$;
- $d_1\nabla_a d_2 \uparrow$, if $d_1 \in A$ or $d_2 \in A$.

**Definition 4.4 (local overlapping).**

1. For nominative data of type $TND_{CA}$ *local overlapping* is a binary operation $\nabla_a^v$ with parameter $v \in V^+$ defined as follows:

$$d_1\nabla_a^v d_2 \cong d_1\nabla_a(\Rightarrow v(d_2)).$$

2. For nominative data of type $TND_{AC}$ *local overlapping* is a binary operation $\nabla_a^v$ parameter $v \in V^+$ defined by induction on the length of the word $v$ as follows:

- if $v \in V$, then $d_1\nabla_a^v d_2 \cong d_1\nabla_a[v \mapsto d_2]$;
- if $v = v_1v_2$, where $v_1 \in V$, $v_2 \in V^+$, $d_1(v_1)\downarrow$, $d_1(v_1) \notin A$, then

$$d_1\nabla_a^v d_2 \cong d_1\nabla_a[v_1 \mapsto d_1(v_1)\nabla_a^{v_2} d_2];$$

- if $v = v_1v_2v_3...v_n$, where $v_i \in V$, and $d_1(v_1)\uparrow$ or $d_1(v_1) \in A$, then

$$d_1 \nabla_a^v d_2 \cong d_1 \nabla_a [v_1 \mapsto [v_2 \mapsto ... \mapsto [v_n \mapsto d_2]...].$$

3. For nominative data of type $TND_{CC}$, *local overlapping* is a binary operation $\nabla_a^v$ with parameter $v \in V^+$ defined as follows:

$$d_1 \nabla_a^v d_2 \cong d_1 \nabla_a (\Rightarrow v(d_2)).$$

**Definition 4.5 (Algebras of nominative data).**

1. An algebra of nominative data of type $TND_{AC}$ is an algebra

$NDA_{AC}(V, A) =< ND(V, A); \{v \Rightarrow_a\}_{v \in V^+}, \{\Rightarrow v\}_{v \in V^+}, \{\nabla_a^v\}_{v \in V^+} >$,

where $ND(V, A)$ is the carrier set, $\{v \Rightarrow_a\}_{v \in V^+}, \{\Rightarrow v\}_{v \in V^+}, \{\nabla_a^v\}_{v \in V^+}$ are operations on $ND(V, A)$.

2. An algebra of nominative data of type $TND_{CA}$ is an algebra

$NDA_{CA}(V, A) =< NDVS(V, A); \{v \Rightarrow_a\}_{v \in V^+}, \{\Rightarrow v\}_{v \in V^+}, \{\nabla_a^v\}_{v \in V^+} >$,

where $NDVS(V, A)$ is the carrier set,

$$\{v \Rightarrow_a\}_{v \in V^+}, \{\Rightarrow v\}_{v \in V^+}, \{\nabla_a^v\}_{v \in V^+}$$

are operations on $NDVS(V, A)$.

3. An algebra of nominative data of type $TND_{CC}$ is an algebra

$NDA_{CC}(V, A) =< NDVC(V, A); \{v \Rightarrow_a\}_{v \in V^+}, \{\Rightarrow v\}_{v \in V^+}, \{\nabla_a^v\}_{v \in V^+} >$,

where $NDVC(V, A)$ is the carrier set,

$$\{v \Rightarrow_a\}_{v \in V^+}, \{\Rightarrow v\}_{v \in V^+}, \{\nabla_a^v\}_{v \in V^+}$$

are operations on $NDVC(V, A)$.

Complex-named data have a hierarchical structure of nomination. However, the information content accessible using associative denomination is identical for some distinct pairs of such data, e.g.

$[v_1 \mapsto [v_2 \mapsto [v_3 \mapsto 1]]]$ and $[v_1 v_2 v_3 \mapsto 1]$ have a different hierarchical structure, but can be considered equivalent since the associative denomination behaves similarly on them.

The following relation on complex-named data, which is called nominative equivalences, formalizes this observation.

**Definition 4.6 (path and terminal path).**

1. *A path* in a complex-named data $d \in NDVC(V, A)$ is a nonempty sequence $(v_1, v_2, ..., v_n)$ of names from $V^+$ such that $((d(v_1))(v_2)...)(v_n)$ is defined. The value $((d(v_1))(v_2)...)(v_n)$ is called the value of the path $(v_1, v_2, ..., v_n)$ in $d$.

2. A terminal path in a complex-named data $d \in NDVC(V, A)$ is a path in $d$ such that its value in $d$ belongs to $A \cup \{\varnothing\}$.

3. A complex-named data $d_1 \in NDVC(V, A)$ is *weakly nominatively included* in a complex-named data $d_2 \in NDVC(V, A)$ (denoted as $d_1 \ll_w d_2$), if either $d_1, d_2 \in A$ and $d_1 = d_2$, or $d_1, d_2 \notin A$ and for each terminal path $(v_1, v_2, ..., v_n)$ in $d_1$ which ends with an atom there exists a terminal path $(v_{1'}, v_{2'}, ..., v_{m'})$ in $d_2$ such that $v_1 v_2 ... v_n = v_{1'} v_{2'} ... v_{m'}$ and the values of $(v_1, v_2, ..., v_n)$ in $d_1$ and of $(v_{1'}, v_{2'}, ..., v_{m'})$ in $d_2$ coincide, and also for each complex name $u \in V^+$ such that $u \Rightarrow_a (d_1)\downarrow \notin W$ the following holds: $u \Rightarrow_a (d_2)\downarrow \notin W$.

4. A complex-named data $d_1 \in NDVC(V, A)$ *is nominatively included* in a complex-named data $d_2 \in NDVC(V, A)$ (denoted as $d_1 \ll d_2$), if either $d_1, d_2 \in A$ and $d_1 = d_2$, or $d_1, d_2 \notin A$ and for each terminal path $(v_1, v_2, ..., v_n)$ in $d_1$ there

exists a terminal path $(v_{1'}, v_{2'}, ..., v_{m'})$ in $d_2$ such that $_c v_1 v_2 ... v_n = v_{1'} v_{2'} ... v_{m'}$ and the value of $(v_1, v_2, ..., v_n)$ in $d_1$ and of $(v_{1'}, v_{2'}, ..., v_{m'})$ in $d_2$ coincide.

5. Complex-named data $d_1, d_2 \in NDVC(V, A)$ are called *nominatively equivalent* (which is denoted as $d_1 \approx d_2$), if $d_1$ is *nominatively* included in $d_2$ , and $d_2$ is *nominatively* included in $d_1$.

The relation of nominative inclusion is a preorder on $NDVC(V, A)$, and nominative equivalence is an equivalence relation on $NDVC(V, A)$.

Nominatively equivalent data can have different hierarchical structures of nomination, but they are equivalent to the same "flat" data of type $TND_{CA}$, e.g.:

- $[v_1 \mapsto [v_2 v_3 \mapsto 1, v_2 v_4 \mapsto 2]] \approx [v_1 v_2 v_3 \mapsto 1, v_1 v_2 v_4 \mapsto 2]$;
- $[v_1 v_2 \mapsto [v_3 \mapsto 1, v_4 \mapsto 2]] \approx [v_1 v_2 v_3 \mapsto 1, v_1 v_2 v_4 \mapsto 2]$.

**Theorem 4.1.** Nominative equivalence is a congruence relation on $NDA_{CC}(V, A)$.

*Proof.* From [NikIv2010] it follows that for each pair $d_1, d_2 \in NDVC(V, A)$ the nomination and associative denomination operations have the following property (called *nominative stability*): if $d_1 \approx d_2$ and the operation is defined on $d_1$, then it is defined on $d_2$ and its values on $d_1$ and $d_2$ are nominatively equivalent. Similarly, from [NikIv2010] it follows that if $v \in V^+$, $d_1, d_2 \notin A$, $d_1 \approx d_{1'}$, $d_2 \approx d_{2'}$, then $d_1 \nabla_a^v d_2 \approx d_{1'} \nabla_a^v d_{2'}$. Then nominative equivalence is a congruence on $NDA_{CC}(V, A)$.

This result allows one to define a quotient algebra of $NDA_{CC}(V, A)$.

Denote as $NDA^{\approx}_{CC}(V,A)$ the quotient algebra of $NDA_{CC}(V,A)$ by the relation of nominative equivalence $\approx$.

**Lemma 4.1.** $NDA_{AC}(V,A)$ is isomorphic to $NDA^{\approx}_{CC}(V,A)$.

*Proof.* Let us define $\iota^{\approx}_{13}: ND(V,A) \to 2^{NDVC(V,A)}$ as follows $\iota^{\approx}_{13}(d) = \{d' \in NDVC(V,A) \mid d' \approx d\}$ (note that $ND(V,A) \subset NDVC(V,A)$). It is easy to check that each nominative equivalence class contains the unique element of $ND(V,A)$ and that the operations on equivalence classes correspond to the operations on elements of $TND_{AC}$. Then $\iota^{\approx}_{13}$ is an isomorphism from $NDA_{AC}(V,A)$ to $NDA^{\approx}_{CC}(V,A)$.

Lemma is proved.

**Lemma 4.2.** $NDA_{CA}(V,A)$ is isomorphic to $NDA^{\approx}_{CC}(V,A)$.

*Proof.* Let $\iota^{\approx}_{23}: NDVS(V,A) \to 2^{NDVC(V,A)}$ be the function: $\iota^{\approx}_{23}(d) = \{d' \in NDVC(V,A) \mid d' \approx d\}$ (note that $NDVS(V,A) \subset NDVC(V,A)$). It is easy to check that each class of nominative equivalence contains the unique element which belongs to $NDVS(V,A)$ and that the operations on equivalence classes correspond to the operations on elements of $TND_{CA}$. Then $\iota^{\approx}_{23}$ is an isomorphism from $NDA_{CA}(V,A)$ to $NDA^{\approx}_{CC}(V,A)$.

Lemma is proved.

**Theorem 4.2.** Algebras $NDA_{AC}(V,A)$, $NDA_{CA}(V,A)$, and $NDA^{\approx}_{CC}(V,A)$ are isomorphic.

*Proof* follows from Lemma 4.1 and Lemma 4.2.

Denote as $D_3/\approx$ the carrier of the algebra $NDA^{\approx}_{CC}(V,A)$.

Let $\iota_{13}^{\approx}$ and $\iota_{23}^{\approx}$ be isomorphisms, defined in the proofs of Lemma 4.1 and Lemma 4.2. For each $d \in D_3$, let $\iota_3^{\approx}(d)$ be the $\approx$-equivalence class of the element $d$. Let us introduce the following inclusion maps for nominative data classes:

$$\iota_{01} : D_0 \rightarrowtail D_1,\ \iota_{02} : D_0 \rightarrowtail D_2,\ \iota_{13} : D_1 \rightarrowtail D_3,\ \iota_{23} : D_2 \rightarrowtail D_3.$$

It is easy to check that the diagram in Fig. 4.1 is commutative.

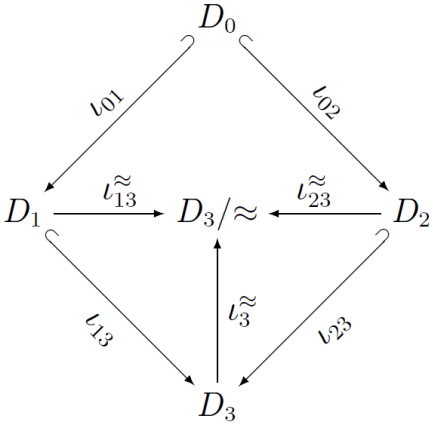



Fig. 4.1. Classes of nominative data and maps between them.

### *4.4. Operational semantics of operations on nominative data*

In this section, we consider the operational semantics of programs over nominative data with complex names and complex values based on a special class of term rewriting systems. This semantics allows one to apply the existing methods applicable to term rewriting systems to the problems of transformation and verification of programs on complex data structures represented using nominative data.

Let $V$ be a fixed set of basic names, $W$ be a set of basic values (atoms). Then $NDVC(V,W)$ is the set of all complex-named data over $V$ and $W$. We will denote as

$$pref(u) = \{v \in V^+ \mid \exists w \in V^* \; u = vw\}$$

the set of nonempty prefixes of a word $u \in V^+$.

Consider the language of programs $SICON_{as}$ (a simple composition-nominative language with associative denomination). Each program in this language is considered as a partial function of the class $NDVC(V,W) \to NDVC(V,W)$,

which can be obtained from the base functions using compositions.

Basic functions are the family of nomination operations $\Rightarrow v$ (for $v \in V^+$), the family of associative denomination operations $v \Rightarrow_a$ (for $v \in V^+$) and the constant function $\bar{\varnothing}$ with the value $\varnothing$.

**Terms and term rewriting systems.** Assume that each function symbol has a corresponding fixed arity.

Let $X = \{x_1, x_2, ...\}$ be an infinite set of variable names.

For each signature $\Sigma$ denote as $T(\Sigma, X)$ the set of terms in the signature $\Sigma$ with variable names from $V$. Then $T(\Sigma, X)$ is the set of *closed* terms.

Let us introduce the notation $t|_p$ for each term $t \in T(\Sigma, X)$ and a finite sequence of positive integers $p = (i_1, ..., i_n)$, $n \geq 0$ as follows:

- $t|_p\uparrow$, if $t \in X$ and $n \geq 1$ or $t = \bar{f}(t_1, ..., t_m)$, where $\bar{f} \in \Sigma$, $m \geq 0$, $n \geq 1$ and $i_1 \notin \overline{1,m}$;
- $t|_p = t$, if $n = 0$;
- $\bar{f}(t_1, ..., t_m)|_p \cong t_{i_1}|_{(i_2,...,i_n)}$, if $m \geq 1$, $n \geq 1$ and $j_1 \in \overline{1,n}$.

The sequences $p$ such that $t|_p\downarrow$ are called *positions* in a term $t$. The empty sequence is called the *root position.*

The expression $t|_p$ is a term which corresponds to the position $p$.

The positions $p_1,...,p_k$ in $t$ are called (pairwise) *incomparable*, if there is no pair of distinct indices $i, j \in \overline{1,k}$ such that one of the positions $p_i$ and $p_j$ is a prefix of another one.

Let us introduce the following notation ($t, t_1, t_2$ are terms):

- $Var(t)$ is the set of variable names which occur in $t$;
- $t_1 \trianglelefteq t_2$ – $t_1$ is a subterm of $t_2$,
  i.e. $t_2 = t_1|_p$ for some position $p$ in $t_1$;
- $t_1 \triangleleft t_2$ – $t_1$ is a strict subterm of $t_2$,
  i.e. $t_1 \trianglelefteq t_2$ and $t_1 \neq t_2$.

A substitution is a total mapping $\sigma : X \to T(\Sigma, X)$ such that $\sigma(x) \neq x$ for a finite subset of elements $x \in X$.

We denote the application of substitution to a term $t$ as $t\sigma$. Note that

- $t\sigma = \sigma(t)$, if $t \in X$,
- $t\sigma = \overline{f}(t_1\sigma, t_2\sigma, ..., t_n\sigma)$, if $t = \overline{f}(t_1,...,t_n)$
  for some $\overline{f} \in \Sigma$, $n \geq 0$ and terms $t_1,...,t_n$.

For partial function $\bar{\sigma} : X \overset{n}{\to} T(\Sigma, X)$, we denote as $ext(\bar{\sigma})$ the substitution which extends $\bar{\sigma}$ on $X$ and satisfies $ext(\bar{\sigma})(x) = x$, if $x \in X$ and $\bar{\sigma}(x)\uparrow$.

Let $t, t_1,...,t_n \in T(\Sigma, X)$. In some cases, we will use the following notation: if $i_1, i_2,..., i_n$ are distinct natural numbers, then $t(t_1 / x_{i_1},...,t_n / x_{i_n})$ denotes the term obtained from $t$ by placing $t_j$ in place of $x_{i_j}$ for $j = \overline{1,n}$.

If $p_1,...,p_n$ are incomparable positions in $t$, we will denote as $t(t_1 / p_1,...,t_n / p_n)$ the term obtained from $t$ by replacing a subterm at the position $p_i$ with $t_i$ for each $i = \overline{1,n}$.

**Definition 4.7.** A constructor-orthogonal term rewriting system (CO TRS) is a triple $R = (\Sigma, C, P)$, where $\Sigma$ is a signature, $C \subseteq \Sigma$ is a subset of *constructors*, and $P$ is a set of rules of the form $l \to r$, $l, r \in T(\Sigma, X)$ such that:

1. Each rule $l \to r \in P$ satisfies the conditions:

- the left-hand side $l$ has a form $\overline{f}(p_1, p_2,..., p_n)$, where $\overline{f} \in \Sigma \setminus C$ is a function symbol, $p_1,..., p_n \in T(C, X)$, $n \geq 0$ are terms constructed from constructors and variable names (*patterns*);
- $Var(r) \subseteq Var(l)$, i.e. variable names on the right-hand side must occur on the left-hand side;
- no variable name occurs in $l$ more than once (*left-linearity*).

2. The following *symbol non-ambiguity* condition is satisfied: in $P$ there are no two distinct rules $l_1 \to r_1$ and $l_2 \to r_2$ such that $l_1\sigma = l_2\sigma'$ for some substitutions $\sigma, \sigma' : X \to T(\Sigma, X)$.

Denote as $\to_R$ the rewrite relation for $R$, i.e. $t_1 \to_R t_2$, where $t_1, t_2 \in T(\Sigma, X)$, if $t_1 |_p = l\sigma$ and $t_2 = t_1(r\sigma / p)$ for some rule $l \to r \in P$, position $p$ in $t_1$ and a substitution $\sigma$. Denote as $\to_R^*$ the reflexive-transitive closure of $\to_R$.

Each CO TRS $R$ is confluent, so each term $t \in T(\Sigma, X)$ has no more than one normal form, (i.e. a term $t'$ such that $t \to_R^* t'$ and there is no $t''$ such that $t' \to_R t''$).

Let us introduce a partial function $nf_R : T(\Sigma, X) \to T(C, X)$:

- $nf_R(t)$ is a normal form of a term $t$, if it exists and belongs to $T(C, X)$;
- $nf_R(t)$ is undefined, otherwise.

CO TRS can be used to give recursive definitions to functions on the initial algebra of the signature $C$. However, here we use TRS to define function over an algebra which models complex-named data.

To define a function on such an algebra using CO TRS, we will introduce an additional unary function symbol $\omega$ which wraps variables which represent atomic values. It can be used in rules in order to distinguish atomic values and non-atomic values. We will also put some restrictions on rules in order to guarantee that the symbol ω is used correctly. Alternatively, order-sorted TRS could be used to define such functions, however, here we do not consider such TRS.

Let us fix a signature of constructors $C$.

Let $F$ be a signature, $F \cap C = \varnothing$, $\omega \notin F \cup C$ is a unary function symbol.

Denote by $C_w = C \cup \{\omega\}$ the extended constructor signature.

Let $\vartheta : T(C \cup F, X) \to T(C_w \cup F, X)$ be a mapping such that $\vartheta(t) = t\sigma$, where $\sigma$ is the substitution $ext(\{x \mapsto \omega(x) \mid x \in Var(t)\})$. This mapping is injective.

Denote by $T_w$ the image of the set $T(C \cup F, X)$ under $\vartheta$. Then the following equality holds: $T_w = \{t\sigma \mid t \in T(C, X)$, $\sigma : X \to \{\omega(x) \mid x \in X\}$ is a substitution $\}$. For example, $c(\omega(x_1), \omega(x_2)) \in T_w^F$, if $c \in C$ is a binary symbol.

Also, denote by $T_w^F = \{t\sigma \mid t \in T(C \cup F, X)$, $\sigma : X \to X \cup \{\omega(x) \mid x \in X\}$ the substitution and there is no $x \in X$ such that $x = \sigma(y)$ and $\omega(x) = \sigma(y')$ for some $y, y' \in X$ $\}$. For example, $f(c(\omega(x_1)), x_2) \in T_w^F$, if $f \in F$ is a binary symbol, and $c \in C$ is a unary symbol.

Note that if $t \in T_w^F$ and $\sigma : X \to T_w$ is a substitution, then $t\sigma \in T_w$.

Denote by $\mathbf{R}$ the class of CO TRS $R = (C_w \cup F, C_w, P)$ such that for each rule $l \to r \in P$:

- if $\omega(t) \trianglelefteq l$ or $\omega(t) \trianglelefteq r$, then $t \in X$;
- if $\omega(x) \trianglelefteq l$, where $x \in X$, then $\omega(x) \trianglelefteq t$ for each $t$ such that $x \triangleleft t$ and $t \trianglelefteq r$;
- if $\omega(x) \trianglelefteq r$, where $x \in X$, then $\omega(x) \trianglelefteq l$.

These conditions imply that if $t_1 \in T_w^F$ and $t_1 \to_R t_2$, then $t_2 \in T_w^F$.

Let us fix a CO TRS $R$ of the class **R**.

**Lemma 4.3.** Let $t \in T(C_w \cup F, X)$. Then $t \in T_w$ if and only if the following conditions hold:

1. if $\omega(t') \trianglelefteq t$, where $t'$ is a term, then $t' \in X$;
2. if $x \in X$, then $\omega(x) \trianglelefteq t'$ for each $t'$ such that $x \triangleleft t'$ and $t' \trianglelefteq t$;
3. $t \notin X$.

*Proof.*

"Only if": assume $t \in T_w$. Then $t = t_0\sigma$ for some $t_0 \in T(C \cup F, X)$ and a substitution $\sigma = ext(\{x \mapsto \omega(x) \mid x \in Var(t_0)\})$. Then from $\omega(t') \trianglelefteq t$ it follows that $\omega(t') \trianglelefteq \sigma(x)$ for some $x \in X$, whence $t' \in X$. Thus the condition 1 holds.

Let $x \triangleleft t'$ and $t' \trianglelefteq t$ for some $x \in X$ and a term $t'$. Then there exists $y \in Var(t_0)$ such that either $x \trianglelefteq \sigma(y) \trianglelefteq t'$, or $x \triangleleft t' \trianglelefteq \sigma(y)$. Then $\sigma(y) = \omega(x)$ and $\omega(x) \trianglelefteq t'$. I.e. the condition 2 holds.

Obviously, the condition 3 also holds.

"If": assume the conditions 1–3. Let $A$ be the set of positions $p$ in $t$ such that $t|_p$ has the form $\omega(t')$ for some $t'$. By the condition 1, for each $p \in A$, there exists (a unique) $x \in X$ which will be denoted by $\phi(p)$ such that $t|_p = \omega(x)$. It is easy to see that the positions in $A$ are pairwise incomparable. Let $p_1, ..., p_n$ be all distinct elements of $A$. Let $t_0 = t(\phi(p_1)/p_1, ..., \phi(p_n)/p_n)$ and $\sigma = ext(\{x \mapsto \omega(x) \mid x \in Var(t_0)\})$. Then $t_0 \in T(C \cup F, X)$. From the conditions 2 and 3 it follows that for each position $p = (i_1, ..., i_k)$

in $t$ such that $t|_p \in X$ it holds $k \geq 1$ and $t|_{(i_1,...,i_{k-1})} = \omega(t|_p)$, so $(i_1,...,i_{k-1}) \in A$, $t_0|_{(i_1,...,i_{k-1})} = t|_p$ and $\sigma(t_0|_{(i_1,...,i_{k-1})}) = \omega(t|_p) = t|_{(i_1,...,i_{k-1})}$. Then $t_0\sigma = t$, thus $t \in T_w$.

Lemma is proved.

**Corollary 4.1**. If $t \in T_w$, $t' \trianglelefteq t$ and $t' \notin X$, then $t' \in T_w$.

**Corollary 4.2**. If $t, t' \in T_w$ and $p$ is a position in $t$ such that $t|_p \notin X$, then $t(t'/p) \in T_w$.

**Lemma 4.4.** If $l \to r \in P$, $\sigma : X \to T(C_w \cup F, X)$ is a substitution and $l\sigma \in T_w$, then $r\sigma \in T_w$.

*Proof.*

For each $x \in Var(l)$ consider the following cases:

- $\omega(x) \trianglelefteq l$. Then $\omega(\sigma(x)) \trianglelefteq l\sigma$ and by Lemma 4.3(1), $\sigma(x) \in X$.
- $\omega(x) \not\trianglelefteq l$. Then $\sigma(x) \trianglelefteq l\sigma$. If $\sigma(x) \in X$, then there exists a term $t' \trianglelefteq l\sigma$ such that $\sigma(x) \triangleleft t'$ and $\omega(\sigma(x)) \not\trianglelefteq t'$, which by Lemma 4.4.1(2) contradicts the assumption $l\sigma \in T_w$. Then $\sigma(x) \notin X$ and by Corollary 1 from Lemma 4.3, we have $\sigma(x) \in T_w$.

Thus, for each $x \in Var(l)$, either $\sigma(x) \in X$, or $\sigma(x) \in T_w$.

Let us check the conditions of Lemma 4.3 for $r\sigma$.

1) Let $\omega(t') \trianglelefteq r\sigma$, where $t'$ is a term. Consider the cases:

- There exists a term $t''$ such that $\omega(t'') \trianglelefteq r$ and $t''\sigma = t'$. Then $t'' \in X$, $\omega(t'') \trianglelefteq l$ and $\sigma(t'') \in X$. Then $t' \in X$.
- There exists $x \in Var(r)$ such that $\sigma(x) = \omega(t')$. Then $x \in Var(l)$ and $\sigma(x) \in T_w$. Then $t' \in X$.

Thus, the condition 1 of Lemma 4.3 holds.

2) Let $x \in X$ and $t'$ is a term such that $x \triangleleft t'$ i $t' \trianglelefteq r\sigma$. Consider the cases:

- There exists a term $t''$ such that $t'' \trianglelefteq r$ and $t''\sigma = t'$. Then $x \in Var(\sigma(y))$ and for some $y \in Var(t'')$. Then

$y \in Var(r) \subseteq Var(l)$. If $\sigma(y) \in X$, then $\omega(y) \trianglelefteq l$ and $x = \sigma(y)$. Then $\omega(x) \trianglelefteq r$ and $\omega(x) \trianglelefteq t'$. If $\sigma(y) \notin X$, then $\sigma(y) \in T_w$, so $\omega(x) \trianglelefteq \sigma(y)$ and $\omega(x) \trianglelefteq t'$.

- There exists $y \in Var(r)$ such that $\sigma(y) = t'$. Then $t' \notin X$, $y \in Var(l)$ and as above, $\sigma(y) \in T_w$. Then $t' \in T_w$, whence $\omega(x) \trianglelefteq t'$.

Thus, the condition 2 of Lemma 4.3 holds.

3) Let us check that $r\sigma \notin X$. If $r\sigma \in X$, then $r \in X$ and $\sigma(r) \in X$. Then $r \in Var(l)$ and $\omega(r) \not\trianglelefteq l$, which contradicts the membership $l\sigma \in T_w$. Thus, the condition 3 of Lemma 4.3 holds.

Thus $r\sigma \in T_w$.

Lemma is proved.

**Lemma 4.5**. If $t_1 \in T_w$ and $t_1 \to_R t_2$, then $t_2 \in T_w$.

*Proof.*

Let $t_1 \in T_w$, $t_1 \to_R t_2$. Then there exists a rule $l \to r \in P$, a position $p$ in $t_1$ and a substitution $\sigma$ such that $t_1|_p = l\sigma$ and $t_2 = t_1(r\sigma / p)$. Since $l\sigma \trianglelefteq t_1$, $l\sigma \notin X$ and $l\sigma \in T_w$, by the Corollary 1 from Lemma 4.4.1, $l\sigma \in T_w$. Then by Lemma 4.4, $r\sigma \in T_w$ and by Corollary 2 from Lemma 4.3, $t_2 \in T_w$.

Lemma is proved.

**Interpretation of terms and function symbols.** Let $\mathbf{D}$ be a free algebra of signature $C$ with the set of generators $W$. The notation $a \in \mathbf{D}$ will denote that $a$ is an element of the carrier of $\mathbf{D}$.

Denote as $\mathbf{D}_*$ the algebra with the carrier $\mathbf{D}$ and the signature $C \cup W$, where the elements of $W$ are considered as constants (0-ary operations).

Denote $T_*^0 = T(C \cup W, \varnothing)$. For each term $t \in T_*^0$ denote as $[t]_0$ (interpretation of $t$) the element of the carrier of $\mathbf{D}$ which is the

value of $t$ under interpretation in $\mathbf{D}_*$. Note that $[t]_0$ is a bijection between $T_*^0$ and the carrier of $\mathbf{D}$.

Let $t \in T_*^0$ and $p_1 < p_2 < ... < p_n$ $(n \geq 0)$ be all positions in the term $t$, at which there are constants from $W$. Denote:

- $ord(t) = n$,
- $\upsilon_s(t) = t(x_{s+1} / p_1, ..., x_{s+n} / p_n) \in T(C, X)$,
- $\beta_s(t)$ is the substitution

$$ext(\{x_{s+j} \mapsto t(p_j) \mid j = \overline{s+1, s+n}\}).$$

Informally, $\upsilon_s(t)$ is a term obtained from $t$ by replacing constants from $W$ with different variables with indices which start with $s$, and the substitution $\beta_s(t)$ allows one to restore the term $t$ from $\upsilon_s(t)$.

Let $R = (C_w \cup F, C_w, P)$ be a CO TRS of class $\mathbf{R}$.

Let us define the interpretation of terms from $T(C_w \cup F, X)$ using $R$.

Let $t \in T(C_w \cup F, X)$, $Var(t) = \{x_{i_1}, ..., x_{i_k}\}$ and $m = i_k$, where $k \geq 0$ and $i_1 < i_2 < ... < i_k$. Denote $[t]_{R,\mathbf{D}}$ (interpretation of $t$) be a partial function $\mathbf{D}^m \to \mathbf{D}$ defined as follows.

Let $D_1, ..., D_m \in \mathbf{D}$. Denote as $t_1, ..., t_m \in T_*^0$ terms such that $D_j = [t_j]_0$, $j = \overline{1, m}$. Let the numbers $s_1, ..., s_m$ be defined as follows: $s_1 = 0$, $s_j = s_{j-1} + ord_B(t_j)$, $j = \overline{1, m}$. Denote

$$t' = t(\upsilon_{s_1}(t_1) / x_1, ..., \upsilon_{s_m}(t_m) / x_m).$$

Note that thanks to the choice of $s_j$, the sets of names of variables occurring in terms $\upsilon_{s_1}(t_1), ..., \upsilon_{s_m}(t_m)$ are pairwise disjoint. Then

- $[t]_{R,\mathbf{D}}(D_1, D_2, ..., D_m) \downarrow = [t'' \beta_{s_1}(t_1) \beta_{s_2}(t_2) ... \beta_{s_m}(t_m)]_0$, if $\vartheta^{-1}(nf_R(\vartheta(t'))) \downarrow = t''$;

- $[t]_{R,\mathbf{D}}(D_1, D_2, ..., D_m)\uparrow$, if $\vartheta^{-1}(nf_R(\vartheta(t')))\uparrow$.

Note that from Lemma 4.5 it follows that if $nf_R(\vartheta(t'))\downarrow$, then $\vartheta^{-1}(nf_R(\vartheta(t')))\downarrow$.

We will mainly consider interpretation not of arbitrary terms from $T(C_w \cup F, X)$, but of terms from the set $T_w^F$.

Let us define the interpretation of function symbols from $F$. For each symbol $\overline{f} \in F$ of arity $m$ let $I_{R,\mathbf{D}}(\overline{f}) = [\overline{f}(x_1, x_2, ..., x_m)]_{R,\mathbf{D}}$ (interpretation of $\overline{f}$).

Let us describe the class of functions which can be defined using CO TRS as follows: let $CTR_{\mathbf{D}}^m$, $m \geq 1$ be the class of partial functions $f : \mathbf{D}^m \to \mathbf{D}$ such that there exists a signature $F$, an $m$-ary symbol $\overline{f} \in F$ and CO TRS $R = (C_w \cup F, C_w, P)$ of class **R** such that $f = I_{R,\mathbf{D}}(\overline{f})$.

We will denote $CTR_{\mathbf{D}} = CTR_{\mathbf{D}}^1$ (unary functions).

**Lemma 4.6**. Let $t_1 \to_R^* t_2$ and $Var(t_1) = Var(t_2)$ for $t_1, t_2 \in T(C_w \cup F, X)$. Then $[t_1]_{R,\mathbf{D}} = [t_2]_{R,\mathbf{D}}$.

*Proof.*

Let $Var(t_1) = \{x_{i_1}, ..., x_{i_k}\}$ and $m = i_k$, where $k \geq 0$ and $i_1 < i_2 < ... < i_k$.

Let $D_1, ..., D_m \in \mathbf{D}$, $t_1, ..., t_m \in T_*^0$ be terms such that $D_j = [t_j]_0$, $j = \overline{1,m}$ and the numbers $s_1, ..., s_m$ be defined as follows: $s_1 = 0$, $s_j = s_{j-1} + ord_B(t_j)$. Denote $t_l' = t_l(\upsilon_{s_1}(t_1)/x_1, ..., \upsilon_{s_m}(t_m)/x_m)$, $l = \overline{1,2}$. Then

$$\vartheta(t_l') = t_l(\upsilon_{s_1}(t_1)/x_1, ..., \upsilon_{s_m}(t_m)/x_m)\sigma = t_l(\upsilon_{s_1}(t_1)\sigma/x_1, ..., \upsilon_{s_m}(t_m)\sigma/x_m)$$

for $l=\overline{1,2}$, where $\sigma = ext(\{x \mapsto \omega(x) \mid x \in \bigcup_{1\le j\le m} Var(\upsilon_{s_j}(t_j))\})$, $j=\overline{1,m}$, since $Var(t_l) \subseteq \{x_1,...,x_m\}$. Then there exists a substitution $\sigma'$ such that $\vartheta(t_l') = t_l\sigma'$, $l=\overline{1,2}$. Since $t_1 \to_R^* t_2$, we have $t_1\sigma' \to_R^* t_2\sigma'$.

If $nf_R(\vartheta(t_1'))\downarrow$, then $nf_R(\vartheta(t_2'))\downarrow = nf_R(\vartheta(t_1'))$ (since $R$ is confluent), so the values $\vartheta^{-1}(nf_R(\vartheta(t_l')))$, $l=\overline{1,2}$ are defined and equal, and $[t_1]_{R,\mathbf{D}}(D_1,D_2,...,D_m) = [t_2]_{R,\mathbf{D}}(D_1,D_2,...,D_m)$.

If $nf_R(\vartheta(t_1'))\uparrow$, then $nf_R(\vartheta(t_2'))\uparrow$, so $[t_1]_{R,\mathbf{D}}(D_1,D_2,...,D_m) \cong [t_2]_{R,\mathbf{D}}(D_1,D_2,...,D_m)$.

Then, since $D_1,...,D_m$ are arbitrary, we have $[t_1]_{R,\mathbf{D}} = [t_2]_{R,\mathbf{D}}$.

Lemma is proved.

**Lemma 4.7.**. If $\overline{c} \in C$ is a function symbol, $c$ is the corresponding operation on $\mathbf{D}$, then $I_{R,\mathbf{D}}(\overline{c}) = c$.

*Proof.*

Let $m$ be the arity of $c$. It is sufficient to show that $[t]_{R,\mathbf{D}} = c$, where $t = \overline{c}(x_1,x_2,...,x_m)$.

Let $D_1,...,D_m \in \mathbf{D}$, $t_1,...,t_m \in T_*^0$ be terms such that $D_j = [t_j]_0$, $j=\overline{1,m}$ and the numbers $s_1,...,s_m$ be defined as $s_1 = 0$, $s_j = s_{j-1} + ord_B(t_j)$, $j=\overline{1,m}$.

Let $t' = t(\upsilon_{s_1}(t_1)/x_1,...,\upsilon_{s_m}(t_m)/x_m) = \overline{c}(\upsilon_{s_1}(t_1),...,\upsilon_{s_m}(t_m))$. Then $t' \in T(C,X)$, so $\vartheta(t') \in T(C_w,X)$, so $nf_R(\vartheta(t')) = \vartheta(t')$. Then

$$[t]_{R,\mathbf{D}}(D_1,D_2,...,D_m)\downarrow = [t'\beta_{s_1}(t_1)\beta_{s_2}(t_2)...\beta_{s_m}(t_m)]_0 =$$
$$= [\overline{c}(t_1,...,t_m)]_0 = c([t_1]_0,...,[t_m]_0) = c(D_1,...,D_m).$$

Since $D_1,...,D_m$ are arbitrary, we have $I_{R,\mathbf{D}}(\overline{c}) = [t]_{R,\mathbf{D}} = c$.

Lemma is proved.

### *4.5. Questions and exercises*

1. Give the definitions of the main types of nominative data.
2. Give the definitions of the operations of nomination, associative denomination and overlapping.
3. Give the definitions of the main compositions of functions over nominative data.
4. What is nominative stability?
5. Give an example of a nominative stable predicate.
6. What is term a rewriting system?

## Chapter 5. Program algebras and logics

### *5.1. The main aspects of programs*

Formalization of programs should include their main aspects.

Traditionally, the *semiotic* aspects (pragmatics, semantics, and syntax) are considered as the main program aspects [Hem2022]. The problem with semiotic aspects consists in weak representation of external aspects (such as social, financial, interactive, efficiency aspect etc.). For this aim only pragmatics can be used. Therefore, this aspect is overloaded with numerous meanings (see, for example [Bjor2009] where several examples of pragmatics are considered).

In [Nik1998] another approach to program aspects was proposed. It is based on the idea of Hegel's development (dialectical) triad:

*thesis – antithesis – synthesis*.

We start with a program triad:

USER – PROBLEM – PROGRAM.

Then we add an execution triad:

PROGRAM – PROCESS OF PROGRAM EXECUTION – USER.

At last, we add program development triad:

PROBLEM – PROGRAM – PROCESS OF PROGRAM DEVELOPMENT.

Finally, we obtain the following pentad of programming notions (Fig. 5.1):

USER – PROBLEM – PROGRAM – PROCESS OF PROGRAM EXECUTION – PROCESS OF PROGRAM DEVELOPMENT

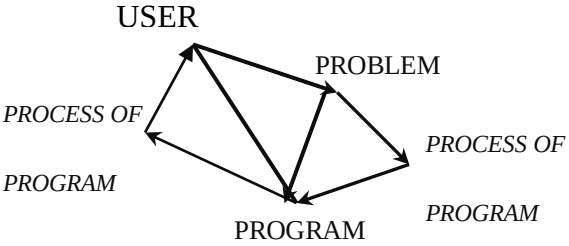



Figure 5.1. Pentad of external program notions.

This pentad specifies the following external program aspects:

- pragmatics – the relation between a user and a program;
- adequacy (correctness) – the relation between a problem and a program;
- computability (executability) – the relation between a program and a process of program execution;
- implicativeness (refinement) – the relation between a program and a process of program development.

External aspects specify such important for software development questions as program correctness with respect to program specification, efficiency of program execution, explication of the process of program development etc. To formulate such questions in a more strict way the internal program aspects should be specified. It means that we should define the main notions specifying internal structure of programs (details are presented in [Nik1998]).

We start with a triad

DATA – FUNCTION – NAME (OF FUNCTION).

Then we add composition triad

FUNCTION – NAME – COMPOSITION

and description triad

NAME – COMPOSITION – DESCRIPTION.

Here DESCRIPTION is regarded as a program text. Thus, we obtained the following pentad of internal program notions (Fig. 5.2):

DATA – FUNCTION – NAME – COMPOSITION – DESCRIPTION.

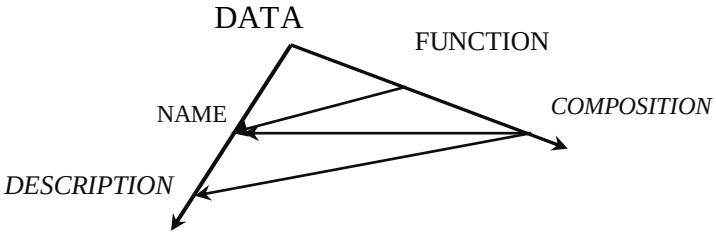



Figure 5.2. Pentad of internal program notions.

This pentad specifies two directions (aspects):

– *semantics*: data – function – composition,
– *syntax*: name – description

and *denotational aspect* relating syntax and semantics.

The presented pentad gives a key for construction of formal program models adopted here.

We formalize semantics by two algebras: data algebra and function (program) algebra. Syntax is formalized by syntactical algebra (often represented by a grammar). We also need to formalize a denotational relation that integrates semantics with syntax. We call this relation a nominative relation because it is based on the following naming relation: name – function. The nominative relation is generalized to the following relation: syntax algebra – program algebra.

The presented scheme of formalizing the main internal program aspects is actually a scheme of denotational semantics [Scott1971]. Such scheme gives the following advantages for program development:

– formality (mathematical rigor);
– descriptive simplicity (short description of the programming languages);
– functional completeness (possibility to program any computable function);

– structural adequacy (the ability to adequately represent the program structure).

The analysis of this scheme demonstrates that it is based on two principles: nominativity and compositionality.

*Nominativity principle* says that programs and their components are constructed with the help of naming relations. *Compositionality principle* states that meaning (semantics) of a program is obtained from meanings of its components with the help of some mapping (composition).

Fundamentality of compositionality principle is widely recognized [Wern2012], as to nominativity principle, it is often treated as important but supplementary.

Here we aim to demonstrate that programs models and methods of program development based on principles of compositionality and nominativity are quite rich and expressive. This approach is called *composition-nominative approach* [Nik1998]. Here we concentrate on representing data processed by programs as nominative data, i.e. as data constructed with the help of naming relation and on construction of algebras and logics over hierarchical nominative data.

Summing up, we can say that the composition-nominative approach aims to formalize software development methods based

1) on methodologically (philosophically) explicated notions of programs and their construction;

2) on a few programming paradigms (compositionality, nominativity, partiality);

3) on strict mathematical notions of program algebras and program logics.

There are different investigations of these topics. Let us indicate some recent publications. At the philosophical level, similar goals are formulated in [Tur2018]; compositionality programming is studied in [Zhang2021], partiality is investigated in [Bove2012]. The advantage of our approach is the use on universal data structures – nominative data – that can represent various data structures used in software development and permits to construct the hierarchy of different algebras and logics generalizing traditional logics.

### *5.2. Example of formalization of a simple program*

In this section we follow the scheme proposed in [Nik-Tym2012a]. We start with considering of a simple language SIPL which is used here to demonstrate how program logics can be constructed. SIPL is similar to such languages as WHILE [Niel1992], IMP [Wins1993], etc.

The grammar of the language is defined as follows:

$s ::= x:=a \mid s_1 ; s_2 \mid$ if $b$ then $s_1$ else $s_2 \mid$while $b$ do $s \mid$ begin $s$ end

$a ::= n \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \mid (a)$

$b ::= a_1 = a_2 \mid a_1 > a_2 \mid b_1 \vee b_2 \mid \neg b \mid (b)$,

where:

- $n$ will range over natural numbers $Nat=\{0, 1, 2, \ldots\}$,
- $x$ will range over variables (names) $V=\{N, M, \ldots\}$,
- $a$ will range over arithmetic expressions *Aex*,
- $b$ will range over Boolean expressions *Bex*,
- $s$ will range over statements (programs) *Stm*.

As an example, consider a SIPL program FACT for calculating the factorial of a given natural number:

*Fact*:=1; while $N > 1$ do begin *Fact*:=*Fact* * $N$; $N$:=$N$–1 end.

Starting from this example we construct several program algebras:

- first, we define denotational semantics of FACT; as a result we obtain an algebra which includes $n$-ary mappings defined on natural numbers;
- then we define a simpler algebra by excluding $n$-ary mappings and introducing simple nominative (quasiary) mappings instead;
- at last, we define a class of quasiary program algebras substituting the set of natural numbers *Nat* by any set $A$. This class is a semantic base for program logics.

We start with denotational semantics [Niel1992, Wins1993] for FACT representing semantic mapping as $[\![.]\!]$. According to this approach, semantics of FACT can be treated as a function over

states of program variables (names). Given a state $d$ in the form [$N \mapsto n$] where $n>0$, this function evaluates new values of *Fact* and *N* such that the value of *Fact* is equal to *n!* and the value of *N* is equal to 1. Analyzing the structure of the program we see that it is constructed from symbols of binary multiplication * and subtraction – operations, binary relation >, natural number 1, variables *N* and *Fact*, and from structuring constructs such as assignment, sequential execution, and loop. Note that subtraction function is partial on *Nat*. To indicate mapping's partiality or totality we will use the sign $\xrightarrow{p}$ for partial mappings and the sign $\xrightarrow{t}$ for total mappings.

Symbols of arithmetic operations and relations represent *n*-ary mappings defined on *Nat*.

For SIPL we define

- *n*-ary functions of type

$$Fn^n_{Nat} = Nat^n \xrightarrow{p} Nat,$$

- *n*-ary predicates of type

$$Pr^n_{Nat} = Nat^n \xrightarrow{p} Bool, \text{ and}$$

- *n*-ary Boolean functions of type

$$Pr^n_{Bool} = Bool^n \xrightarrow{p} Bool, n \geq 0.$$

Here *Bool* denotes the set of Boolean values.

We use the same notation for language symbols and mappings they represent. Thus, we can write for SIPL that +, –, $*$: $Fn^2_{Nat}$, $\vee$: $Pr^2_{Bool}$, $\neg$ : $Pr^1_{Bool}$, =, >: $Pr^2_{Nat}$. A natural number *n* can be considered as a null-ary function denoted $\overline{n}$.

For presenting semantics of arithmetic and Boolean expressions we first define the set of states $ST = V \xrightarrow{p} Nat$ (denoted here as ${}^{V}Nat$). States may be considered as nominative data with simple names and simple values (*SS*-data). Mappings over nominative data are called *nominative mappings*, because they are defined on *ST* constructed with the help of naming (nominative) relation. Arithmetic expressions denote nominative functions (called *quasiary functions in this case*) of the set

$$Fn^V_{Nat} = ST \xrightarrow{p} Nat = {}^V Nat \xrightarrow{p} Nat;$$

therefore $[\![a]\!] \in Fn^V_{Nat}$ for $a \in Aex$. Boolean expressions denote nominative predicates (called *quasiary predicates*) of the set

$$Pr^V_{Nat} = {}^V Nat \xrightarrow{p} Bool; \text{ thus } [\![b]\!] \in Pr^V_{Nat} \text{ for } b \in Bex.$$

Semantics of variables in arithmetic expressions are defined by a parametric denomination (denomination) function $'x$: $Fn^V_{Nat}$. The function $'x$ returns the value of the variable $x$ in a given state. For example, denomination functions that yield values of names *Fact* and *N* are denoted as $'Fact$ and $'N$ respectively. These values may not be defined, so denomination functions are partial.

The semantics of structuring constructs is defined with the help of special operators called *compositions*. To define semantics of structured arithmetic and Boolean expressions, say of expressions *Fact*N* and *N*>1 respectively, we will use superposition compositions which are a semantic explication of term substitution.

Superposition $S^n_F : Fn^n_{Nat} \times (Fn^V_{Nat})^n \xrightarrow{t} Fn^V_{Nat}$ of nominative functions $g_1,\ldots, g_n$ into an $n$-ary function $f^n$ is an operator such that

$$S^n_F(f^n, g_1,\ldots, g_n)(d) = f^n\ (g_1(d),\ldots,g_n(d))$$

where $d$ is a state.

The same formula can be used for defining superposition

$$S^n_P :\ Pr^n_{Nat} \times (Fn^V_{Nat})^n \xrightarrow{t} Pr^n_{Nat}$$

of nominative functions $g_1,\ldots, g_n$ into $n$-ary predicate $p^n$, and for superperposition $S^n_B :\ Pr^n_{Bool} \times (Pr^V_{Nat})^n \xrightarrow{t} Pr^V_{Nat}$ of nominative predicates $p_1,\ldots, p_n$ into $n$-ary Boolean function $f^n_B$. Thus,

$$[\![Fact*N]\!] = S^2_F(*, 'Fact, 'N),\ \ [\![N>1]\!] = S^2_P(>, 'N, \bar{1}).$$

Programs from *Stm* denote bi-nominative (bi-quasiary) functions of the class $Prg^V_{Nat} = ST \xrightarrow{p} ST = {}^V Nat \xrightarrow{p} {}^V Nat$.

Semantics of structured statements is defined by the following compositions with conventional meaning:

– assignment $AS^x$: $Fn^n_{Nat} \xrightarrow{t} Prg^V_{Nat}$ ($x$ is a parameter from $V$);

– sequential execution •: $Prg^V_{Nat} \times Prg^V_{Nat} \xrightarrow{t} Prg^V_{Nat}$;

– conditional $IF$: $Pr^V_{Nat} \times Prg^V_{Nat} \times Prg^V_{Nat} \xrightarrow{t} Prg^V_{Nat}$;

– loop $WH$: $Pr^V_{Nat} \times Prg^V_{Nat} \xrightarrow{t} Prg^V_{Nat}$.

For example,

$[\![ Fact:=Fact*N ]\!] = AS^{Fact}(S^2_F(*, 'Fact, 'N))$,

$[\![ Fact:=Fact*N;\ N:=N–1 ]\!] = AS^{Fact}(S^2_F(*, 'Fact, 'N)) \bullet AS^N(S^2_F(–, 'N, \overline{1}))$.

Note, that for • we commute arguments of conventional functional composition, thus, $f \bullet g = g \circ f$.

So, our example leads to the following program algebra which has 7 carriers:

$\mathfrak{Al}^V_{Nat} = < Fn^2_{Nat}, Pr^2_{Bool}, Pr^1_{Bool}, Pr^2_{Nat}, Fn^V_{Nat}, Pr^V_{Nat}, Prg^V_{Nat};$

$\overline{n}, +, –, *, =, >, \vee, \neg, S^2_F, S^2_P, S^2_B, S^1_B, 'x, AS^x, \bullet, IF, WH >$.

It is important to note that semantics of SIPL program expressions can be represented as terms of this algebra. This simplifies investigations of SIPL programs because the constructed algebra completely specifies their semantics.

The term for FACT in $\mathfrak{Al}^V_{Nat}$ is as follows:

$$AS^{Fact}(\overline{1}) \bullet WH(S^2_P(>, 'N, \overline{1}),$$
$$AS^{Fact}(S^2_F(*, 'Fact, 'N)) \bullet AS^N(S^2_F(–, 'N, \overline{1})).$$

Let us admit that this term and its components can denote partial mappings because the denomination functions can be undefined; also, subtraction and loop can lead to undefinedness.

Having defined algebra $\mathfrak{Al}^V_{Nat}$ as formalism for presenting program semantics, we can use it to study properties of programs. Such properties can be employed in program reasoning. Several examples are presented in [NikSh2013].

Algebra $\mathfrak{An}^{V}_{Nat}$ is based on natural numbers, but it is clear that we can consider other data types such as *integer*, *real*, etc. So, we can ask a question: what SIPL properties of programs remain valid under type variations?

To clarify this question, we should split the set of operation symbols of $\mathfrak{An}^{V}_{Nat}$ into two parts:

- logical symbols, which are relatively type-independent, and
- non-logical (or descriptive), which depend upon specifics of the carriers.

Inspecting definitions of our algebra we can see that symbols $\overline{n}$, $+$, $-$, $*$, $=$, $>$ are descriptive symbols since they are defined over natural numbers. Other symbols may be treated as logical.

Now we can go to the next step of program algebra development: we exclude *n*-ary functions and predicates and concentrate on logical symbols.

Still, *n*-ary functions and predicates can be represented as nominative mappings. Indeed, a pair $(n_1, n_2)$ can be represented as a state $[1 \mapsto n_1,\ 2 \mapsto n_2]$, where 1 and 2 are treated as standard variables (names). Instead of standard names we can use names from *V*. For instance, binary multiplication function * can be represented as a parametric quasiary function ***x*y*** (printed in bold font) such that ***x*y*** $= S^2_F(*, 'x, 'y)$; here *x* and *y* are parameters from *V*. In this case an arithmetic expression *Fact*N* will be represented by quasiary function ***Fact*N***. We can also use natural numbers in such representations.

So, for our language SIPL we introduce parametric quasiary functions ***x+y***, ***x–y***, and ***x*y*** instead of binary functions; also we introduce new parametric quasiary predicates ***x=y*** and ***x>y*** instead of relations (with *x* and *y* as parameters).

Thus, every *n*-ary function can be represented as a parametric quasiary function. But in this case to represent semantics of expressions we should introduce special superpositions $S^{v_1,\ldots,v_n}_F$ (or $S^{\bar{v}}_F$)

and $S_P^{v_1,\ldots,v_n}$ (or $S_P^{\bar{v}}$), which are called *superpositions into quasiary function* and *predicate* respectively (names $v_1,\ldots, v_n$ are disticnt names, deliting $|_{-v_1,\ldots,v_n}$ and $\nabla$ updating operations will be defined in the next sections):

$S_F^{v_1,\ldots,v_n}(f^q, g_1,\ldots, g_n)(d) = f^q(d\,|_{-v_1,\ldots,v_n}\nabla[v_1\mapsto g_1(d),\ldots,v_n\mapsto g_n(d)])$ and

$S_P^{v_1,\ldots v_n}(p^q, g_1,\ldots, g_n)(d) = p^q(d\,\|_{-v_1,\ldots,v_n}\nabla[v_1\mapsto g_1(d),\ldots,v_n\mapsto g_n(d)])$,

where $f^q\in Fn_{Nat}^V$, $p^q\in Pr_{Nat}^V$.

These formulas mean that we first delete in $d$ the components with names $v_1,\ldots, v_n$ and then add to the obtained data components with names $v_1,\ldots, v_n$ and values $g_1(d), \ldots, g_n(d)$ respectively if they are defined; after that we evaluate the value of a function $f^q$ or predicate $p^q$ on the obtained state. In this case semantics of expressions is represented with this superposition, say, expression *Fact**($N$–1) can be represented as $S_F^M$ (***Fact*M***, ***N*–1**).

As to logical symbols $\vee: Bool^2 \xrightarrow{t} Bool$ and $\neg: Bool \xrightarrow{t} Bool$, it is better to treat them as binary compositions over quasiary predicates (denoted by the same signs) $\vee: Pr_{Nat}^V \times Pr_{Nat}^V \xrightarrow{t} Pr_{Nat}^V$ and $\neg: Pr_{Nat}^V \xrightarrow{t} Pr_{Nat}^V$. Such representations also give possibility to work with partial predicates. Actually, it means that our compositions are similar to Kleene's strong connectives [Kleene1952].

### *5.3. Program algebra of quasiary mappings*

At this point we arrived to the program algebra of quasiary mappings with constants:

$\mathfrak{Alc}_{Nat}^V = < Fn_{Nat}^V, Pr_{Nat}^V, Prg_{Nat}^V;$

$\bar{n}$, ***x–y***, ***x+y***, ***x*y***; ***x=y***, ***x>y***, $\vee$, $\neg$, $S_F^{\bar{v}}$, $S_P^{\bar{v}}$, $'x$, $AS^x$, •, *IF*, *WH*> .

Here we have 6 parametric descriptive symbols considered as algebra's constants, and 9 logical symbols. All logical symbols are treat-

ed as compositions (denomination functions is a null-ary composition).

In $\mathfrak{Alc}^V_{Nat}$ semantics of FACT is represented by the following term:

$$AS^{Fact}(\bar{1}) \bullet WH(\boldsymbol{N>1}, AS^{Fact}(\boldsymbol{Fact*N}) \bullet AS^{N}(\boldsymbol{N–1})).$$

To make our algebra more general we eliminate descriptive symbols $\bar{n}$ **, *x–y, x+y, x*y, x=y, x>y*** introducing into signature two sets *Fs* and *Ps* of function and predicate symbols that denote any quasiary function or predicate.

Being interested in general laws of reasoning about programs, we make the next step and define compositions for any set *A* of basic values. First, we define classes of quasiary functions, predicates and program functions by the following formulas respectively:

$$Fn^V_A = {}^V A \xrightarrow{p} A,\ Pr^V_A = {}^V A \xrightarrow{p} Bool,\ Prg^V_A = {}^V A \xrightarrow{p} {}^V A.$$

Then, we redefine compositions defined earlier over the class of natural numbers *Nat* for set *A*. This leads to the following definition.

**Definition 5.1.** A tuple

$$\mathfrak{Alq}^V_A = < Fn^V_A,\ Pr^V_A,\ Prg^V_A; \vee, \neg, S^{\bar{v}}_F,\ S^{\bar{v}}_P, 'x,\ AS^x, \bullet, IF, WH >$$

is called *a program algebra of quasiary mappings*.

Thus, we have constructed a class of program algebras (for various *A*), representing program semantics for languages with different domains. Such algebras form the semantic base for program logics. They may be also called *algorithmic algebras*.

We can distinguish several levels for such algebras (Fig. 5.3).

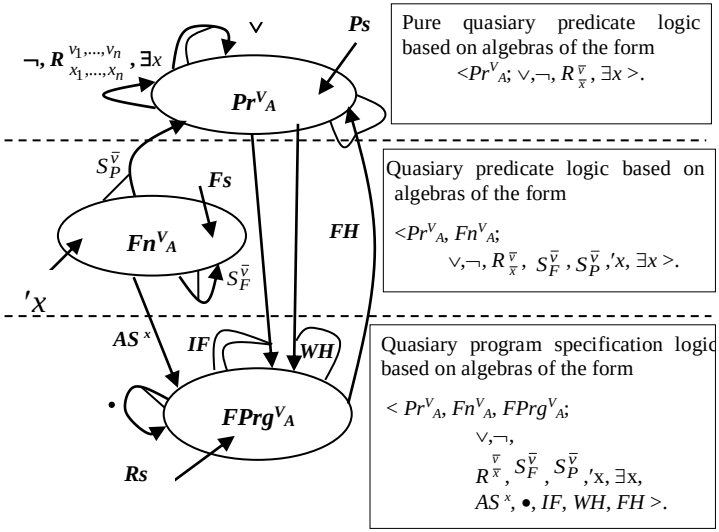



Figure 5.3. Diagram of levels of program specification algebra of quasiary mappings.

## *5.4. Program specification algebra of quasiary mappings*

To use such algebras for program specification, we should add more powerful compositions: monotone Floyd-Hoare composition *FH* [KrNik2013], composition of existential quantification $\exists x \ (x \in V)$, variable assignment predicate $Ex(x \in V)$, and composition of predicate complement $\sim$. The latter composition is required for constructing sound rules of specification program logic of Floyd–Hoare type. Also, we add identity function *id*.

In the following definitions we assume that $p, q \in Pr_A^V$, $prg \in Prg_A^V$, and $d, r \in {}^V A$; $p(d)\downarrow$ means that $p$ is defined on data $d$, $p(d)\downarrow = r$ means that $p$ is defined on $d$ with a value $r$, $p(d)\uparrow$ means that $p$ is undefined on $d$.

**Definition 5.2.** *Monotone Floyd-Hoare composition* is a mapping $FN : Pr_A^V \times Prg_A^V \times Pr_A^V \to Pr_A^V$ such that

$$FH(p, prg, q)(d) = \begin{cases} T, & \text{if } q(prg(d))\downarrow = T \text{ or } p(d)\downarrow = F, \\ F, & \text{if } p(d)\downarrow = T \text{ and } q(prg(d))\downarrow = F, \\ & \text{undefined in other cases.} \end{cases}$$

**Definition 5.3.** *Parametric composition of existential quantification* with a parameter $x \in V$ is a mapping $\exists x : Pr_A^V \to Pr_A^V$ such that

$$\exists x p(d) = \begin{cases} T, & \text{if there exists } b \in A \text{ such that } p(d\nabla[x \mapsto b])\downarrow = T, \\ F, & \text{if } p(d\nabla[x \mapsto a])\downarrow = F \text{ for all } a \in A, \\ & \text{undefined in all other cases.} \end{cases}$$

Here the updating operation $\nabla$ is defined as follows: if $d_1$ and $d_2$ are nominative sets then $d = d_1 \nabla d_2$ consists of all named pairs of $d_2$ and only those pairs of $d_1$, whose names are not defined in $d_2$. A shorter form for $d\nabla[x \mapsto a]$ is $d\nabla x \mapsto a$.

**Definition 5.4.** *Composition of disjunction* is a mapping $\vee : Pr_A^V \times Pr_A^V \to Pr_A^V$ such that

$$(p \vee q)(d) = \begin{cases} T, & \text{if } p(d)\downarrow = T \text{ or } q(d)\downarrow = T, \\ F, & \text{if } p(d)\downarrow = F \text{ and } q(d)\downarrow = F, \\ & \text{undefined in all other cases.} \end{cases}$$

**Definition 5.5.** *Composition of negation is a mapping* $\neg : Pr_A^V \to Pr_A^V$ such that $(\neg p)(d) = \begin{cases} T, & \text{if } p(d)\downarrow = F, \\ F, & \text{if } p(d)\downarrow = T, \\ \text{undefined, if } & p(d)\uparrow. \end{cases}$

**Definition 5.6.** *Composition of predicate complement is a mapping* $\sim : Pr_A^V \to Pr_A^V$ such that

$$(\sim p)(d) = \begin{cases} T, & \text{if } p(d) \text{ is undefined,} \\ \text{undefined, if } & p(d) \text{ is defined.} \end{cases}$$

**Definition 5.7.** *Nullary parametric variable assignment composition (predicate)* with a parameter $x \in V$ is a mapping $Ex : Pr_A^V$ such that

$$(Ex)(d) = \begin{cases} T, & \text{if } d(x) \text{ is defined,} \\ F, & \text{if } d(x) \text{ is undefined.} \end{cases}$$

**Definition 5.8.** *Program specification algebra* over quasiary mappings is a tuple

$\mathfrak{As}_A^V = < Pr_A^V, Fn_A^V, Prg_A^V; \vee, \neg, {}'x, S_F^{\bar{v}}, S_P^{\bar{v}}, \exists x, Ex, \sim, id, AS^x, \bullet,$ *IF, WH, FH*>

that is extension of algebra $\mathfrak{Aq}_A^V$ with the compositions defined above.

In this algebra we can define renomination composition $R_{x_1,\ldots,x_n}^{v_1,\ldots,v_n}$ as derived from superposition composition in the following way:

$$R_{x_1,\ldots,x_n}^{v_1,\ldots,v_n}(p) = S_P^{v_1,\ldots,v_n}(p, {}'x_1, \ldots, {}'x_n).$$

The class of such algebras forms the semantic base for quite powerful Floyd–Hoare-like logics of quasiary mappings.

It is important to admit that restricting $\mathfrak{As}_A^V$ on one and two carriers we obtain the following algebras (similar to those presented in Fig. 5.3):

– propositional algebra $< Pr_A^V; \vee, \neg >$;

– quantifier-free pure predicate algebra $< Pr_A^V; \vee, \neg, R_{x_1,\ldots,x_n}^{v_1,\ldots,v_n} >$;

– first-order pure predicate algebra $< Pr_A^V; \vee, \neg, R_{x_1,\ldots,x_n}^{v_1,\ldots,v_n}, \exists x >$;

– first-order predicate algebra (with functions)

$$< Pr_A^V, Fn_A^V; \vee, \neg, {}'x, S_F^{\bar{v}}, S_P^{\bar{v}}, \exists x >;$$

– extended first-order predicate algebra

$$< Pr_A^V, Fn_A^V; \vee, \neg, {}'x, S_F^{\bar{v}}, S_P^{\bar{v}}, \exists x, Ex, \sim >.$$

Such algebras were studied in our previous papers.

Based on classes of considered algebras, various propositional, first-order, and composition-nominative program and specification logics can be defined. We can go even further and define modal and temporal composition-nominative logics.

Summing up, we would like to say that

- semantics of simple programs can be presented by terms of program algebras with compositions as operations of this algebra;
- program, functions, and predicates are defined on simple nominative data;
- program logics can be constructed directly on program algebras by extending their signatures with special "logical" compositions.

In the sequel we consider composition-nominative program specification algebras and logics based on mappings over hierarchical nominative data defined in Chapter 4.

## *5.5. Program specification logic over hierarchical nominative data*

To define a logic $L^H$ we should define

– class of algebras which form semantic base for logic;
– logic language (based on logic signature);
– class of interpretations of language constructs;
– consequence relation;
– inference relation based on some calculus.

**Language of $L^H$**

Let *Ps* be a set of predicate symbols, *Fs* be a set of function symbols, *Prgs* be a set of program symbols, *V* be an infinite set of names (variables). Usually, within V an infinite subset $V_U$ of unessential variables is identified.

**Definition 5.9.** A tuple

$$S = (V, V_U; \vee, \neg, S_P^{\bar{u}}, S_F^{\bar{u}}, 'x, \exists x, Ex, \sim, AS^x, \bullet, IF, WH, FH, id; Ps, Fs, Prgs)$$

is called the *language signature*.

For simplicity, we use the same notation for symbols of compositions and compositions themselves.

**Definition 5.10.** Given signature *S*, the *language* of $L^H$ which consists of the set of *terms* $Tr^S$, *predicate formulas* $Fr^S$, *program*

*texts* $Pt^S$ , *and* the set of *Floyd-Hoare program assertions* $FH^S$ is define as follows.
Terms are defined inductively in the following way:

– if $F \in Fs$ then $F \in Tr^S$ ;

–if $x \in V^+$ then $'x \in Tr^S$ ;

–if $(u_1,...,u_n) \in \bar{U}(V), t, t_1,...,t_n \in Tr^S$ then $S_F^{u_1,...,u_n}(t,t_1,...,t_n) \in Tr^S$ .

Formulas are defined inductively in the following way:

–if $P \in Ps$ then $P \in Fr^S$ ;

–if $\Phi, \Psi \in Fr^S$ , $x \in V^+$ then $\Phi \vee \Psi, \neg\Phi, \sim\Phi, \exists x, Ex \in Fr^S$ ;

–if $(u_1,...,u_n) \in \bar{U}(V), t_1,...,t_n \in Tr^S, \Phi \in Fr^S$ then

$S_P^{u_1,...,u_n}(\Phi, t_1,...,t_n) \in Fr^S$ .

Programs are defined inductively in the following way:

–if $prs \in Prgs$ then $prs \in Pt^S$ ;

–if $x \in V^+, prg_1, prg_2 \in Pt^S, \Phi \in Fr^S$ then

$$id, AS^x, prg_1 \bullet prg_2, IF(\Phi, prg_1, prg_2), WH(\Phi, prg_1) \in Pt^S .$$

Floyd-Hoare assertions are defined inductively in the following way:

– if $prg \in Pt^S, pre, post \in Fr^S$ then $\{pre\} prg \{post\} \in FH^S$ .

### $L^H$-interpretations

Let *S* be a signature and A be a set of basic values, $\mathfrak{Alhs}_A^V$ be a program specification algebra over hierarchical nominative data.

**Definition 5.11.** An interpretation in $L^H$ is a tuple $J = (\mathfrak{Alhs}_A^V, I_{Fs}, I_{Ps}, I_{Prgs})$ where

– $I_{Fs} : Fs \xrightarrow{t} FnH_A^V$ ,

– $I_{Ps} : Ps \xrightarrow{t} PrH_A^V$ ,

– $I_{Prgs} : Prgs \xrightarrow{t} PrgH_A^V$.

We will also demand that mappings $I_{Fs}, I_{Ps}, I_{Prgs}$ agree with the set of unessential variables from $V_U$.

Given an interpretation $J = (\mathfrak{Alhs}_A^V, I_{Fs}, I_{Ps}, I_{Prgs})$ we define mappings

$$I_{Tr} : Tr^S \xrightarrow{t} FnH_A^V, I_{Fr} : Fr^S \xrightarrow{t} PrH_A^V, I_{Pt} : Pt^S \xrightarrow{t} PrgH_A^V$$

as the natural extensions (by structural induction) of $I_{Fs}, I_{Ps}, I_{Prgs}$ to $Tr^S, Fr^S, Pt^S$ respectively.

Also, we define interpretation mapping of Floyd-Hoare assertions $J_{FH} : FH^S \xrightarrow{t} PrH_A^V$ as follows:

$$J_{FH}(\Phi, prg, \Psi) = FH(J_{Fr}(\Phi), J_{Pt}(prg), J_{Fr}(\Psi)).$$

For any $t \in Tr^S, \Phi \in Fr^S$, $prg \in Pt^S$, $fha \in FH^S$ we denote by $t_J$, $\Phi_J$, $prg_J$, $fha_J$ their interpretations in *J*.

Formula Φ is *irrefutable in J* if predicate $\Phi_J$ is irrefutable. Formula Φ is *irrefutable* if Φ is irrefutable for any interpretation *J*. Irrefutability may be treated as *partial validity*.

Variable *x* is *unessential for* Φ if for any *J* variable *x* is unessential for $\Phi_J$ (denoted $x \updownarrow \Phi$). Variable *x* is *unessential for* $\Gamma \subseteq Fr$ if for any *J* variable *x* is unessential for any formula $\Phi \in \Gamma$. The set of all variables (names) that occur in Φ is denoted *nm*(Φ). In the same way we define unessential variables for terms and programs. The set $fu(\Phi) = V_U \setminus nm(\Phi)$ is called the set of *fresh unessential* variables for Φ.

For any $\Gamma \subseteq Fr$ we define

$$nm(\Gamma) = \bigcup_{\Phi \in \Gamma} nm(\Phi) \text{ and } fu(\Phi) = \bigcap_{\Phi \in \Gamma} fu(\Phi).$$

We generalize notation *fu*(Γ) on sequences of formulas and sets of formulas. Given formula Φ and interpretation *J* the truth, falsity, and undefinedness domains of $\Phi_J$ are denoted $tt(\Phi_J)$, $ff(\Phi_J)$, and $\perp(\Phi_J)$ respectively. These notations are generalized for a set of formulas $\Sigma \subseteq Fr^S$:

$$tt^{\cap}(\Sigma_J) = \bigcap_{\Phi\in\Sigma} tt(\Phi_J)\,,\ \ ff^{\cap}(\Sigma_J) = \bigcap_{\Phi\in\Sigma} ff(\Phi_J)\,,\ \perp^{\cap}(\Sigma_J) = \bigcap_{\Phi\in\Sigma} \perp(\Phi_J)\,.$$

**Logical consequence relation under conditions of undefinedness**

Composition $\sim$ of predicate complement introduces undefinedness domains into consideration therefore more complicated consequence relation is required. Here we will use new consequence relation between sets of formulas denoted $|=_{IR}^{\perp}$ [NikMam2019].

**Definition 5.12.** Let $\Gamma, U, \Delta \subseteq Fr^S$. Then

$\Delta$ *is irrefutable consequence of* $\Gamma$ *under undefinedness conditions* U *in interpretation J* (denoted $U/\Gamma_J |=_{IR}^{\perp} \Delta$ ) *if*

$$tt^{\cap}(\Gamma_J) \cap \perp^{\cap}(U_J) \cap ff^{\cap}(\Delta_J) = \varnothing;$$

$\Delta$ is *logical irrefutable consequence of* $\Gamma$ *under undefinedness conditions* U (denoted $U/\Gamma |=_{IR}^{\perp} \Delta$) if $U/\Gamma_J |=_{IR}^{\perp} \Delta$ for any interpretation *J*.

Note, that for $U = \varnothing$, we obtain traditional *logical irrefutability relation* $\Gamma |=_{IR} \Delta$.

Let us introduce on $Fr^S$ binary relation $\simeq$ of logical strong equality. Namely, $\Phi \simeq \Psi$ if $\Phi_J = \Psi_J$ for any interpretation *J*.

Now we describe the main properties of $|=_{IR}^{\perp}$.

**Theorem 5.1.** Let $\Phi \simeq \Psi$. Then:

$U/\Phi, \Gamma |=_{IR}^{\perp} \Delta \Leftrightarrow U/\Psi, \Gamma |=_{IR}^{\perp} \Delta;$

$U/\Gamma |=_{IR}^{\perp} \Delta, \Phi \Leftrightarrow U/\Gamma |=_{IR}^{\perp} \Delta, \Psi;$

$U, \Phi/\Gamma |=_{IR}^{\perp} \Delta \Leftrightarrow U, \Psi/\Gamma |=_{IR}^{\perp} \Delta.$

For $|=_{IR}^{\perp}$ the following properties of formula decomposition hold.

**Theorem 5.2.** For any $U, \Gamma, \Delta \subseteq Fr$, $\Phi, \Psi, \vartheta \in Fr$:

$\vee_L$) $U/\Phi\vee\Psi, \Gamma |=_{IR}^{\perp} \Delta \ \Leftrightarrow\ U/\Phi, \Gamma |=_{IR}^{\perp} \Delta$ and $U/\Psi, \Gamma |=_{IR}^{\perp} \Delta$;

$\vee_R$) $U/\Gamma |=_{IR}^{\perp} \Delta, \Phi\vee\Psi \ \Leftrightarrow\ U/\Gamma |=_{IR}^{\perp} \Delta, \Phi, \Psi$;

$\vee_U$) $U,\Phi\vee\vartheta/\Gamma |=_{IR}^{\perp} \Delta \ \Leftrightarrow\ U,\Phi,\vartheta/\Gamma |=_{IR}^{\perp} \Delta$ and $U,\Phi/\Gamma |=_{IR}^{\perp} \vartheta, \Delta$ and

$$U,\vartheta/\Gamma |=_{IR}^{\perp} \Phi, \Delta;$$

$\neg_L$) $U/\neg\Phi, \Gamma |=_{IR}^{\perp} \Delta \ \Leftrightarrow\ U/\Gamma |=_{IR}^{\perp} \Delta, \Phi$;

$\neg_R$) $U/\Gamma \mid\!=_{IR}^{\perp} \Delta, \neg\Phi \;\Leftrightarrow\; U/\Phi,\Gamma \mid\!=_{IR}^{\perp} \Delta$;

$\neg_U$) $U,\neg\vartheta/\Gamma \mid\!=_{IR}^{\perp} \Delta \;\Leftrightarrow\; U,\vartheta/\Gamma \mid\!=_{IR}^{\perp} \Delta$;

$\sim_U$) $U,\sim\Phi/\Gamma \mid\!=_{IR}^{\perp} \Delta \;\Leftrightarrow\; U/\Phi,\Gamma \mid\!=_{IR}^{\perp} \Delta$ and $U/\Gamma \mid\!=_{IR}^{\perp} \Delta,\Phi$;

$\sim_L$) $U/\sim\Phi,\Gamma \mid\!=_{IR}^{\perp} \Delta \;\Leftrightarrow\; U,\Phi/\Gamma \mid\!=_{IR}^{\perp} \Delta$.

More properties of irrefutability consequence relation are presented in [NikMam2019]. Here we consider only properties related to superposition and quantification compositions which use complex names.

Semantic properties induce three corresponding properties for $\mid\!=_{IR}^{\perp}$, depending on the position of a formula (in the left side of $\mid\!=_{IR}^{\perp}$, in the right side of $\mid\!=_{IR}^{\perp}$, in the undefinedness conditions of $\mid\!=_{IR}^{\perp}$). Such properties are formulated in a similar way, for example, the following properties $S\vee_L$, $S\vee_R$, $S\vee_U$ and $S\exists_L$, $S\exists_R$, $S\exists_U$ are induced by $S\vee$ and $S\exists$:

$S\vee_L$) $U/S_P^{\bar{v}}(\Phi\vee\Psi,\bar{t}),\Gamma \mid\!=_{IR}^{\perp} \Delta \;\Leftrightarrow\; U/S_P^{\bar{v}}(\Phi,\bar{t})\vee S_P^{\bar{v}}(\Psi,\bar{t}),\Gamma \mid\!=_{IR}^{\perp} \Delta$;

$S\vee_R$) $U/\Gamma \mid\!=_{IR}^{\perp} \Delta, S_P^{\bar{v}}(\Phi\vee\Psi,\bar{t}) \;\Leftrightarrow\; U/\Gamma \mid\!=_{IR}^{\perp} \Delta, S_P^{\bar{v}}(\Phi,\bar{t})\vee S_P^{\bar{v}}(\Psi,\bar{t})$;

$S\vee_U$) $U, S_P^{\bar{v}}(\Phi\vee\Psi,\bar{t})/\Gamma \mid\!=_{IR}^{\perp} \Delta \;\Leftrightarrow\; U, S_P^{\bar{v}}(\Phi,\bar{t})\vee S_P^{\bar{v}}(\Psi,\bar{t})/\Gamma \mid\!=_{IR}^{\perp} \Delta$;

$S\exists_L$) $U/\; S_P^{\bar{v}}(\exists x\Phi,\bar{t}),\Gamma \mid\!=_{IR}^{\perp} \Delta \;\Leftrightarrow\; U/\; \exists z\, S_P^{\bar{v},x}(\Phi,\bar{t},'z),\Gamma \mid\!=_{IR}^{\perp} \Delta$;

$S\exists_R$) $U/\Gamma \mid\!=_{IR}^{\perp} \Delta,\; S_P^{\bar{v}}(\exists x\Phi,\bar{t}) \;\Leftrightarrow\; U/\Gamma \mid\!=_{IR}^{\perp} \Delta,\; \exists z\, S_P^{\bar{v},x}(\Phi,\bar{t},'z)$;

$S\exists_U$) $U,\; S_P^{\bar{v}}(\exists x\Phi,\bar{t})\; /\Gamma \mid\!=_{IR}^{\perp} \Delta \;\Leftrightarrow\; U/\; \exists z\, S_P^{\bar{v},x}(\Phi,\bar{t},'z),\Gamma \mid\!=_{IR}^{\perp} \Delta$.

For properties $S\exists_L$, $S\exists_R$, $S\exists_U$ the following restrictions should hold:

$$z \div \bar{v},\; z \updownarrow \Phi \text{ and } z \updownarrow \bar{t}.$$

(names should be incomparable)

### *5.6. Questions and exercises*

1. What are the main aspects of programs?

2. Describe the pentad of the main programming notions?
3. Describe the pentad of the main notions of program?
4. What are the semantics, syntax and pragmatics of programs?
5. Formulate the principle of compositionality?
6. Formulate the principle of nominativity?
7. How to formalize programs?
8. What types of programs can be described by algebras of quasiary mappings?
9. What are the types of superposition compositions?
10. What are the main program compositions?
11. How is the algebra of specifications of quasiary mappings defined?
12. How are program compositions defined?
13. What are the main operations in the algebra of nominative data?
14. How is the algebra of specifications over hierarchical nominative data defined?
15. How is the language of the logic of specifications over hierarchical nominative data defined?
16. How are interpretations defined in the logic of specifications over hierarchical nominative data? What is the consequence relation in the logic of specifications over hierarchical nominative data?
17. What are the properties of the consequence relation in the logic of specifications over hierarchical nominative data?

## Chapter 6. Deductive systems for compositional logics

Deductive systems aim to determine the relations of inference.

Among the different types of such systems, we distinguish Gentzen-type systems, in particular, sequent systems (sequent calculi).

These systems have a number of advantages:

- they are deductively strong, i.e. many other deductive systems can be considered as special cases of Gentzen-type systems;
- they are intuitively simple;
- they have the subformula property, i.e. in the process of proof only subformulas of the original formulas or their variants are used;
- they are in some sense isomorphic to the semantics of compositions, namely, knowing the semantics of compositions, one can construct derivation (inference) rules for these compositions, and vice versa, knowing the semantics of compositions, one can construct their sequential rules.

We consider sequent calculi for the logics considered in the previous chapters.

### *6.1. Sequent calculus for classical propositional logic*

For the propositional logic the truth table method allows to check the truth of a formula or set of formulas. This method belongs to the class of semantic methods, that is, it is based on the evaluation of the values of formulas (semantics of formulas). Another class of methods is syntactic methods, which are based on syntactic transformations of formulas or sets of formulas. It should be noted that such syntactic transformations are induced by the semantic properties of formulas, so semantic and syntactic methods are related to

each other. Let us consider sequent methods, in particular, sequent calculus of propositional logic.

The idea of sequent methods is very simple. Consider the *consequence statement* $A_1, A_2, \ldots, A_k \models B$. By definition, it will be true if and only if for all values of the propositional variables the truth of $A_1, A_2, \ldots, A_k$ implies the truth of $B$.

Let us prove the truth of the statement by the method of contradiction. To do this, let us assume that the consequence statement is falsifiable (is refutable), that is, there is a set of values of the propositional variables such that $A_1, A_2, \ldots, A_k$ are true, and $B$ is false. If in the process of transformations we reach a contradiction, that is, we obtain a formula that is simultaneously true and false, then our assumption will be refuted. Therefore, the consequence statement is true.

The idea of finding a contradiction is the basic of the sequent method. Let us consider a more general form of the consequence statement: $A_1, A_2, \ldots, A_k \models B_1, B_2, \ldots, B_m$. For sequent methods, such statements are written as $A_1, A_2, \ldots, A_k \rightarrow B_1, B_2, \ldots, B_m$ and are called sequents. Here, the symbol $\rightarrow$ is a new symbol that does not belong to the object language. It should be understood that this symbol is a metasymbol, but in terms of the meaning of the introduced logical consequence, it is actually an implication symbol. We call such sequents as bivalent sequents.

To explain the sequent method, let us consider a simple example, namely: we will prove the truth of the consequence statement $\neg(A \vee B) \models \neg A \wedge \neg B$. Let us assume that this statement is refutable, that is, there is such a set of values of the propositional variables that the formula $\neg(A \vee B)$ is true and the formula $\neg A \wedge \neg B$ is false. We will write these assumptions in the form ${}_1\neg(A \vee B)$, ${}_0\neg A \wedge \neg B$ that is, we will index (mark) true formulas with the sign 1, and false ones with the sign 0. We will obtain two indexed formulas that form a marked sequent.

Note that each binary sequence (sequence of the form A1, A2, …, Ak |= B1, B2, …, Bm ) is uniquely (with preservation of formula

structures) translated into a semantically equivalent marked bivalent sequent, and vice versa, each bivalent marked sequent is uniquely translated (with preservation of formulas) into a semantically equivalent binary sequent. Semantic equivalence means the coincidence of semantic values, i.e. the identity of the corresponding binary functions.

Next, let us try to simplify the formulas of this sequent. The main operation of the first formula is the negation. Therefore, $\neg(A \vee B)$ will be true if and only if $A \vee B$ is false. Thus, the initial sequent is transformed into a new sequent ${}_0A \vee B$, ${}_0\neg A \wedge \neg B$. Next, we will simplify the formula ${}_0A \vee B$, which must be false. The main operation of this formula is disjunction.

A disjunction will be false if and only if its arguments are simultaneously false. In our case, the arguments of the disjunction are $A$ and $B$. Therefore, the indexed formula ${}_0A \vee B$ can be replaced by two indexed formulas ${}_0A$ and ${}_0B$. Thus, the sequent ${}_0A \vee B$, ${}_0\neg A \wedge \neg B$ has transformed into a new sequent ${}_0A$, ${}_0B$, ${}_0\neg A \wedge \neg B$. In the last sequent, there is only one composite (non-atomic) formula ${}_0\neg A \wedge \neg B$, which indicates that the formula $\neg A \wedge \neg B$ must be false.

A conjunction has the value false in three cases, namely:

1) when the first and second arguments of the conjunction are false;

2) when the first argument is false, and the second is true;

3) when the first argument is true, and the second is false.

These three cases can be replaced by two simplified cases:

1) the first argument of the conjunction is false;

2) the second argument of the conjunction is false.

Therefore, further for the conjunction (as for the disjunction and implication) we will consider only two cases. Thus, the formula $\neg A \wedge \neg B$ will be false if and only if:

1) $\neg A$ is false or

2) $\neg B$ is false.

This means that the sequent ${}_0A, {}_0B, {}_0\neg A \wedge \neg B$ has turned into two sequents: ${}_0A, {}_0B, {}_0\neg A$ and ${}_0A, {}_0B, {}_0\neg B$. In the first of them, the formula ${}_0\neg A$ is non-atomic, which we replace with ${}_1A$. We obtain the sequent ${}_0A, {}_0B, {}_1A$. Analyzing it, we see that $A$ must be true and false at the same time, since the sequent includes the indexed formulas ${}_0A$ and ${}_1A$. We have obtained a contradiction; therefore, this sequent cannot refute our initial statement. Now let's analyze the second sequent ${}_0A, {}_0B, {}_0\neg B$. Transforming ${}_0\neg B$ to ${}_1B$, we obtain the sequent ${}_0A, {}_0B, {}_1B$. In this sequent, a contradiction arises for the formula $B$. Therefore, this sequent cannot refute the initial consequence statement.

Sequents of this type, that is, those containing a formula indexed as 1 and 0, are called *closed*. We denote such contradictory formulas with the sign ×. We will also denote closed sequent with this same sign.

We have considered all possible cases and demonstrated that the consequence statement cannot be refuted, therefore it is true.

The considered example allows us to draw the following general conclusions:

1) to verify the truth of the consequence statement, we can use the method of proof by contradiction, which consists in finding a contradiction for the sets of indexed (labeled, marked) formulas obtained in the process of proof;

2) the transformation of indexed formulas occurs according to rules induced by the semantic properties of operations; the rules have premises and conclusions;

3) sequent proof (sequent derivation) can be presented in the form of a tree, its root is the initial marked sequent, and the transitions are given by sequent rules;

4) for a sequent tree there are three possible cases

– the tree is closed (all branches are closed). In this case the initial consequence statement is logically true, and the corresponding formula is a tautology,

– there is one unclosed branch in the tree. In this case the initial consequence statement is falsifiable (refuted),

– all branches in the tree are unclosed. In this case the initial consequence is a contradiction.

How to formulate sequent rules (forms)? We see that for each logical operation (composition) there are two rules:

- the first specifies the transformation of the formula marked with 1,
- the second – the formula marked with 0.

For example, for the disjunction operation, two such rules can be written (the premises are written above the line, the conclusions are written below the line):

$$\frac{{}_1A \qquad {}_1B}{{}_1A \vee B} \text{ and } \frac{{}_0A,\ {}_0B}{{}_0A \vee B}.$$

These rules are formulated for one formula, but the sequent can also have other formulas. Let us denote their set as $\Sigma$. The general rule will look like this (on the left side we write the name of the rule):

$${}_1\neg\ \frac{{}_0A, \Sigma}{{}_1\neg A, \Sigma} \qquad {}_0\neg\ \frac{{}_1A, \Sigma}{{}_0\neg A, \Sigma}.$$

Note that different sequents can be separated by the alternative sign "|", but we do not do this, knowing that the absence of a comma indicates the place where two sequents are separated.

Similarly, rules can be constructed for other logical operations. We get the following set of sequent rules:

$${}_1\neg\ \frac{{}_0A, \Sigma}{{}_1\neg A, \Sigma} \qquad {}_0\neg\ \frac{{}_1A, \Sigma}{{}_0\neg A, \Sigma}$$

$${}_1\vee\ \frac{{}_1A, \Sigma \qquad {}_1B, \Sigma}{{}_1A \vee B, \Sigma} \qquad {}_0\vee\ \frac{{}_0A,\ {}_0B, \Sigma}{{}_0A \vee B, \Sigma}$$

$$1\wedge\ \frac{{}_1A,{}_1B,\Sigma}{{}_1A\wedge B,\Sigma} \qquad 0\wedge\ \frac{{}_0A,\Sigma\quad {}_0B,\Sigma}{{}_0A\wedge B,\Sigma}$$

$$1\to\ \frac{{}_0A,\Sigma\qquad {}_1B,\Sigma}{{}_1A\to B,\Sigma} \qquad 0\to\ \frac{{}_1A,\ {}_0B,\Sigma}{{}_0A\to B,\Sigma}$$

$$1\sim\ \frac{{}_0A,\ {}_0B,\Sigma\qquad {}_1A,{}_1B,\Sigma}{{}_1A\leftrightarrow B,\Sigma} \qquad 0\sim\ \frac{{}_0A,{}_1B,\Sigma\qquad {}_1A,{}_0B,\Sigma}{{}_0A\leftrightarrow B,\Sigma}$$

When performing derivation tasks, it is advisable to write a proof in the reverse order – that is, write the conclusions above the line, and the premises – below the line. In other words, we write more complex formulas above the line, and simpler ones – below the line. Therefore, in the future we will write sequent rules in the following form:

$$1\neg\ \frac{{}_1\neg A,\Sigma}{{}_0A,\Sigma} \qquad 0\neg\ \frac{{}_0\neg A,\Sigma}{{}_1A,\Sigma}$$

$$1\vee\ \frac{{}_1A\vee B,\Sigma}{{}_1A,\Sigma\qquad {}_1B,\Sigma} \qquad 0\vee\ \frac{{}_0A\vee B,\Sigma}{{}_0A,\ {}_0B,\Sigma}$$

$$1\wedge\ \frac{{}_1A\wedge B,\Sigma}{{}_1A,{}_1B,\Sigma} \qquad 0\wedge\ \frac{{}_0A\wedge B,\Sigma}{{}_0A,\Sigma\quad {}_0B,\Sigma}$$

$$1\to\ \frac{{}_1A\to B,\Sigma}{{}_0A,\Sigma\qquad {}_1B,\Sigma} \qquad 0\to\ \frac{{}_0A\to B,\Sigma}{{}_1A,\ {}_0B,\Sigma}$$

$$1\sim\ \frac{{}_1A\leftrightarrow B,\Sigma}{{}_0A,\ {}_0B,\Sigma\qquad {}_1A,{}_1B,\Sigma} \qquad 0\sim\ \frac{{}_0A\leftrightarrow B,\Sigma}{{}_0A,{}_1B,\Sigma\qquad {}_1A,{}_0B,\Sigma}$$

Direct and reverse (invers) writing of rules can be interpreted as direct and reverse proof. Therefore, for reverse proof, we can write premises under the dash in the rule, and conclusions above the dash. Please pay attention to this when constructing proofs.

Below is the sequent derivation of the considered consequence statement $\neg(A\vee B)\models\neg A\wedge\neg B$ (actually the proof gives inference statement $\neg(A\vee B)\vdash\ \neg A\wedge\neg B$):

$$(_1\neg)\dfrac{{}_1\neg(A\vee B),\ {}_0\neg A\wedge\neg B}{(_0\vee)\dfrac{{}_0A\vee B,\ {}_0\neg A\wedge\neg B}{(_0\wedge)\dfrac{{}_0A,\ {}_0B,\ {}_0\neg A\wedge\neg B}{(_1\neg)\dfrac{{}_0A,\ {}_0B,\ {}_0\neg A}{\dfrac{{}_0A_{\times},\ {}_0B,\ {}_1A_{\times}}{\times}}\qquad\dfrac{{}_0A,\ {}_0B,\ {}_0\neg B}{\dfrac{{}_0A,\ {}_0B_{\times},\ {}_1B_{\times}}{\times}}}}}$$

**Example 6.1.** The antecedent as the consequent (the premise as the conclusion): $A\to B\to A$. We recall that the brackets in this formula are arranged as follows: $(A\to(B\to A))$, that is, the main operation is the first implication. We present the formula as a marked sequent: ${}_0A\to B\to A$. We apply the rule ${}_0\to$ and write the name of the rule to the left of the dash. We obtain

$$(_0\to)\dfrac{{}_0A\to B\to A}{{}_1A,\ {}_0B\to A}$$

Now we analyze the obtained sequent ${}_1A,\ {}_0B\to A$. It has only one complex formula ${}_0B\to A$. We index this formula with the * sign and apply the rule ${}_0\to$ to it.

$$(_0\to)\dfrac{{}_0A\to B\to A}{(_0\to)\dfrac{{}_1A,\ {}_0B\to A*}{{}_1A,\ {}_1B,\ {}_0A}}$$

The last sequent is closed, since it contains the formulas ${}_1A$ and ${}_0A$, that is, the sequent demands that $A$ is both true and false. We index such formulas with the sign $\times$. This is a contradiction, so the initial sequent has no refutation. We denote such a closed sequent with the sign $\times$. Finally, the sequent derivation tree has the form

$$(_0\to)\dfrac{{}_0A\to B\to A}{(_0\to)\dfrac{{}_1A,\ {}_0B\to A*}{\dfrac{{}_1A_{\times},\ {}_1B,\ {}_0A_{\times}}{\times}}}$$

Having constructed a truth table, we verify that the initial formula is a tautology:

| $A\ B$ | $B\to A$ | $A\to B\to A$ |
|---|---|---|
| **0 0** | **1** | **1** |
| **0 1** | **0** | **1** |
| **1 0** | **1** | **1** |
| **1 1** | **1** | **1** |

**Example 6.2.** Commutativity of antecedents (premises):

$$A\to(B\to C)\sim B\to(A\to C).$$

We form a marked sequent: ${}_0A\to(B\to C)\sim B\to(A\to C)$. Its main operation is the equivalence ~, so we apply the rule ${}_0\sim$.

$$({}_0\sim)\frac{{}_0A\to(B\to C)\sim B\to(A\to C)}{{}_0A\to(B\to C),{}_1B\to(A\to C)\quad {}_1A\to(B\to C),{}_0B\to(A\to C)}$$

We have obtained two new sequents. Let's consider the first sequent

$${}_0A\to(B\to C),\ {}_1B\to(A\to C).$$

It is advisable to apply the rule ${}_0\to$ to the first formula, which is indexed by the sign *. We also apply this same rule to the second sequent:

$$({}_0\to)\frac{{}_0A\to(B\to C)^*,\ {}_1B\to(A\to C)}{({}_0\to)\dfrac{{}_1A,\ {}_0B\to C^*,\ {}_1B\to(A\to C)}{{}_1A,\ {}_1B,\ {}_0C,\ {}_1B\to(A\to C)}}$$

In the derived sequent, the only complex formula is the formula ${}_1B\to(A\to C)$, to which we apply the rule ${}_1\to$. We apply the same rule again to the formula ${}_1A\to C$. All the resulting sequents will be closed:

$$(_1\to)\frac{{}_1A,\ {}_1B,\ {}_0C,\ \ {}_1B\to(A\to C)*}{\dfrac{{}_1A,\ {}_1B_\times,\ {}_0C,\ {}_0B_\times}{\times}\qquad (_1\to)\dfrac{{}_1A,\ {}_1B,\ {}_0C,\ {}_1A\to C*}{\dfrac{{}_1A_\times,\ {}_1B,\ {}_0C,\ {}_0A_\times}{\times}\qquad\dfrac{{}_1A,\ {}_1B,\ {}_0C_\times,\ {}_1C_\times}{\times}}}$$

It remains to consider the sequent

$${}_1A\to(B\to C),\ \ {}_0B\to(A\to C)$$

We apply the rule ${}_0\to$ twice to the indicated formulas:

$$(_0\to)\frac{{}_1A\to(B\to C),\ \ {}_0B\to(A\to C)*}{(_0\to)\dfrac{{}_1A\to(B\to C),\ \ {}_1B,\ {}_0(A\to C)*}{{}_1A\to(B\to C)*,\ \ {}_1B,\ {}_1A,\ {}_0C}}$$

We build a derivation tree for the last obtained sequent:

$$(_1\to)\frac{{}_1A\to(B\to C)*,\ \ {}_1B,\ {}_1A,\ {}_0C}{\dfrac{{}_0A_\times,\ \ {}_1B,\ {}_1A_\times,\ {}_0C}{\times}\qquad (_1\to)\dfrac{{}_1B\to C*,\ \ {}_1B,\ {}_1A,\ {}_0C}{\dfrac{{}_0B_\times,\ \ {}_1B_\times,\ {}_1A,\ {}_0C}{\times}\qquad\dfrac{{}_1C_\times,\ \ {}_1B,\ {}_1A,\ {}_0C_\times}{\times}}}$$

All branches of the derivation tree are closed, therefore the initial formula is a tautology. This can also be verified by constructing a semantic table (truth table):

| $A\,B\,C$ | $B\to C$ | $A\to C$ | $A\to(B\to C)$ | $B\to(A\to C)$ | $A\to(B\to C)\sim$ $\sim B\to(A\to C)$ |
|---|---|---|---|---|---|
| **0 0 0** | **1** | **1** | **1** | **1** | **1** |
| **0 0 1** | **1** | **1** | **1** | **1** | **1** |
| **0 1 0** | **0** | **1** | **1** | **1** | **1** |
| **0 1 1** | **1** | **1** | **1** | **1** | **1** |
| **1 0 0** | **1** | **0** | **1** | **1** | **1** |
| **1 0 1** | **1** | **1** | **1** | **1** | **1** |
| **1 1 0** | **0** | **0** | **0** | **0** | **1** |
| **1 1 1** | **1** | **1** | **1** | **1** | **1** |

**Example 6.3.** Conjunction of antecedents (premises):

$$A \to B \to C \sim A \wedge B \to C .$$

Parentheses can be arranged in this way: $(A \to (B \to C)) \sim ((A \wedge B) \to C)$, therefore the main operation is equivalence. We form a marked sequent and apply rule ${}_0\sim$. We obtain

$$({}_0\sim)\frac{{}_0A \to B \to C \sim A \wedge B \to C}{{}_0A \to B \to C,\ {}_1B \to C \quad {}_1A \to B \to C,\ {}_0A \wedge B \to C}$$

We will construct a derivation tree separately for the two specified sequents. Let's start with the first sequent:

$$({}_0\to)\frac{{}_0A \to B \to C^*,\ {}_1B \to C}{({}_0\to)\dfrac{{}_1A,\ {}_0B \to C^*,\ {}_1B \to C}{({}_1\to)\dfrac{{}_1A,\ {}_1B,\ {}_0C,\ {}_1B \to C^*}{\underline{{}_1A,\ {}_1B,\ {}_0C,\ {}_0B}\ \times \qquad \underline{{}_1A,\ {}_1B,\ {}_0C,\ {}_1C}\ \times}}}$$

All branches of the inference tree are closed, that is, the sequent is inferred (this is indicated by $|\!-_0 A \to B \to C_*,\ {}_1B \to C$). Now we build the inference tree for the second sequent:

$$({}_0\to)\frac{{}_1A \to B \to C,\ {}_0A \wedge B \to C^*}{({}_1\wedge)\dfrac{{}_1A \to B \to C,\ {}_1A \wedge B^*,\ {}_0C}{({}_1\to)\dfrac{{}_1A \to B \to C^*,\ {}_1A,\ {}_1B,\ {}_0C}{\underline{{}_0A,\ {}_1A,\ {}_1B,\ {}_0C}\ \times \qquad ({}_1\to)\dfrac{{}_1B \to C^*,\ {}_1A,\ {}_1B,\ {}_0C}{\underline{{}_0B,\ {}_1A,\ {}_1B,\ {}_0C}\ \times \qquad \underline{{}_1C,\ {}_1A,\ {}_1B,\ {}_0C}\ \times}}}}$$

All branches are closed, so $A \to B \to C \sim A \wedge B \to C$ is a tautology.

This can also be checked by building a semantic table:

| $A\,B\,C$ | $B \to C$ | $A \wedge B$ | $A \to (B \to C)$ | $(A \wedge B) \to C$ | $A \to (B \to C) \sim$ $\sim (A \wedge B) \to C$ |
|---|---|---|---|---|---|
| **0 0 0** | **1** | **0** | **1** | **1** | **1** |
| **0 0 1** | **1** | **0** | **1** | **1** | **1** |
| **0 1 0** | **0** | **0** | **1** | **1** | **1** |
| **0 1 1** | **1** | **0** | **1** | **1** | **1** |
| **1 0 0** | **1** | **0** | **1** | **1** | **1** |
| **1 0 1** | **1** | **0** | **1** | **1** | **1** |
| **1 1 0** | **0** | **1** | **0** | **0** | **1** |
| **1 1 1** | **1** | **1** | **1** | **1** | **1** |

**Example 6.4.** Non-associativity of implication: check an equivalence

$$A \to B \to C \sim (A \to B) \to C\,.$$

**Remark.** Implication is a right-associative operation, therefore $A \to B \to C = A \to (B \to C)$; the main operation is equivalence ~, i.e. the brackets can be arranged as follows:

$$(A \to (B \to C)) \sim ((A \to B) \to C)$$

To prove the truth of the initial formula, we need to prove (deduce) the marked implication:

$$_0A \to (B \to C) \sim (A \to B) \to C$$

The main operation is equivalence. Therefore, we apply the rule $_0\sim$. We obtain the derivation

$$_0\sim \frac{_0A \to (B \to C) \sim (A \to B) \to C}{_0A \to (B \to C),\ _1(A \to B) \to C \quad _1A \to (B \to C),\ _0(A \to B) \to C}$$

Since the full derivation does not fit on the width of the page, we will consider the derivation for two sequents separately: $_0A \to (B \to C),\ _1(A \to B) \to C$ and $_1A \to (B \to C),\ _0(A \to B) \to C$.

Let's analyze the sequent ${}_0A \to (B \to C),\ {}_1(A \to B) \to C$. In this sequent, we should pay attention to the formula ${}_0A \to (B \to C)$. The rule for its transformation is quite simple, therefore, in the sequent we will apply ${}_0\to$ for the specified formula. To indicate such an application, we index the specified formula with the sign *.

We got the derivation

$$({}_0\to)\frac{{}_0A \to (B \to C)^*,\ {}_1(A \to B) \to C}{{}_1A,\ {}_0(B \to C),\ {}_1(A \to B) \to C}$$

Analyzing the obtained sequent, we see that the simplest transformation will be for the formula ${}_0(B \to C)$. We index this formula in the sequent with the * sign and apply the rule ${}_0\to$. The derivation takes the form

$$({}_0\to)\frac{{}_0A \to (B \to C)^*,\ {}_1(A \to B) \to C}{({}_0\to)\dfrac{{}_1A,\ {}_0(B \to C)^*,\ {}_1(A \to B) \to C}{{}_1A,\ {}_1B,\ {}_0C,\ {}_1(A \to B) \to C}}$$

Now the only formula to which the sequent rule can be applied is the formula ${}_1(A \to B) \to C$. We get

$$({}_0\to)\frac{{}_0A \to (B \to C)^*,\ {}_1(A \to B) \to C}{({}_0\to)\dfrac{{}_1A,\ {}_0(B \to C)^*,\ {}_1(A \to B) \to C}{({}_1\to)\dfrac{{}_1A,\ {}_1B,\ {}_0C,\ {}_1(A \to B) \to C^*}{{}_1A,\ {}_1B,\ {}_0C,\ {}_0A \to B \quad {}_1A,\ {}_1B,\ {}_0C,\ {}_1C}}}$$

We got two new sequents. The second sequent is closed, since it contains ${}_0C$ and ${}_1C$, that is, the sequent requires that $C$ to be both true and false. We index these formulas with the sign $\times$. We got a contradiction, therefore, on this branch of the proof, the initial sequent has no refutation. We denote such a closed sequent by the sign $\times$.

We transform the first sequent, applying the rule ${}_0\to$:

$$(_0\to)\dfrac{{}_0A\to(B\to C)^*,\ {}_1(A\to B)\to C}{(_0\to)\dfrac{{}_1A,\ {}_0(B\to C)^*,\ {}_1(A\to B)\to C}{(_1\to)\dfrac{{}_1A,\ {}_1B,\ {}_0C,\ {}_1(A\to B)\to C^*}{(_0\to)\dfrac{{}_1A,\ {}_1B,\ {}_0C,\ {}_0A\to B^*}{\dfrac{{}_1A,\ {}_1B_\times,\ {}_0C,\ {}_1A,\ {}_0B_\times}{\times}}\qquad\dfrac{{}_1A,\ {}_1B,\ {}_0C_\times,\ {}_1C_\times}{\times}}}}$$

The obtained sequents are closed. So, all cases have been considered, the sequent ${}_0A\to(B\to C),\ {}_1(A\to B)\to C$ has been proved.

Now let's start building the derivation of the sequent ${}_1A\to(B\to C),\ {}_0(A\to B)\to C$. Its analysis shows that it is advisable to transform the formula ${}_0(A\to B)\to C$. We have

$$(_0\to)\dfrac{{}_1A\to(B\to C),\ {}_0(A\to B)\to C^*}{{}_1A\to(B\to C),\ {}_1A\to B,\ {}_0C}$$

Transforming the first formula of the obtained sequent, we have:

$$(_0\to)\dfrac{{}_1A\to(B\to C),\ {}_0(A\to B)\to C^*}{(_1\to)\dfrac{{}_1A\to(B\to C)^*,\ {}_1A\to B,\ {}_0C}{{}_0A,\ {}_1A\to B,\ {}_0C\qquad {}_1B\to C,\ {}_1A\to B,\ {}_0C}}$$

We have obtained two sequents: ${}_0A,\ {}_1A\to B,\ {}_0C$ and ${}_1B\to C,\ {}_1A\to B,\ {}_0C$. We apply the rule ${}_0\to$ to the first one. We obtain

$$(_0\to)\dfrac{{}_1A\to(B\to C),\ {}_0(A\to B)\to C^*}{(_1\to)\dfrac{{}_1A\to(B\to C)^*,\ {}_1A\to B,\ {}_0C}{(_1\to)\dfrac{{}_0A,\ {}_1A\to B^*,\ {}_0C}{\dfrac{{}_0A,\ {}_0A,\ {}_0C}{!}\qquad\dfrac{{}_0A,\ {}_1B,\ {}_0C}{!}}\qquad {}_1B\to C,\ {}_1A\to B,\ {}_0C}}$$

This case gives two unclosed sequents: ${}_0A,\ {}_0A,\ {}_0C$ and ${}_0A,\ {}_1B,\ {}_0C$, which we denote by the sign !. An unclosed sequent provides a counterexample, that is, it indicates the values of the propositional variables that refute the formula.

Indeed, we calculate the values $A\to(B\to C)\leftrightarrow(A\to B)\to C$ for the values 0, 0, 0, respectively. We have:

$$\mathbf{0\to(0\to0)\sim(0\to0)\to0=0\to1\sim1\to0=1\to0=0}.$$

We obtain the same result with the values 0, 1, 0 for $A$, $B$, $C$ respectively.

Now we build a derivation tree for the sequent ${}_1B \to C,\ {}_1A \to B,\ {}_0C$ having values 0, 1, 0 for $A$, $B$, $C$ respectively. We have

$$\mathbf{0 \to (1 \to 0) \sim (0 \to 1) \to 0 = 0 \to 0 \sim 1 \to 0 = 1 \to 0 = 0}.$$

Let us return to the construction of the inference tree for the sequent ${}_1B \to C,\ {}_1A \to B,\ {}_0C$. We obtain

$$(_1\to)\ \frac{{}_1B \to C^*,\ {}_1A \to B,\ {}_0C}{(_1\to)\ \dfrac{{}_0B,\ {}_1A \to B^*,\ {}_0C}{\dfrac{{}_0B,\ {}_0A,\ {}_0C}{!} \quad \dfrac{{}_0B_\times,\ {}_1B_\times,\ {}_0C}{\times}} \qquad (_1\to)\ \dfrac{{}_0C,\ {}_1A \to B^*,\ {}_0C}{\dfrac{{}_0C,\ {}_0A,\ {}_0C}{!} \quad \dfrac{{}_0C,\ {}_1B,\ {}_0C}{!}}}$$

The sequents ${}_0B,\ {}_0A,\ {}_0C$ ; ${}_0C,\ {}_0A,\ {}_0C$ ; ${}_0C,\ {}_1B,\ {}_0C$ are unclosed. They all provide counterexamples for the initial sequent.
Let's build a truth table for it:

| $A\,B\,C$ | $B \to C$ | $A \to B$ | $A \to (B \to C)$ | $(A \to B) \to C$ | $A \to (B \to C) \sim$ $\sim (A \to B) \to C$ |
|---|---|---|---|---|---|
| **0 0 0** | **1** | **1** | **1** | **0** | **0** |
| **0 0 1** | **1** | **1** | **1** | **1** | **1** |
| **0 1 0** | **0** | **1** | **1** | **0** | **0** |
| **0 1 1** | **1** | **1** | **1** | **1** | **1** |
| **1 0 0** | **1** | **0** | **1** | **1** | **1** |
| **1 0 1** | **1** | **0** | **1** | **1** | **1** |
| **1 1 0** | **0** | **1** | **0** | **0** | **0** |
| **1 1 1** | **1** | **1** | **1** | **1** | **0** |

**Example 6.5.** Prove the consequence statement $a \vee b, \neg a \vee c \models b \vee c$.

We form a marked sequent: ${}_1a \vee b,\ {}_1\neg a \vee c,\ {}_0b \vee c$. We build its sequent derivation:

$$(_0\vee)\ \frac{{}_1a \vee b,\ {}_1\neg a \vee c,\ {}_0b \vee c^*}{(_1\vee)\ \dfrac{{}_1a \vee b^*,\ {}_1\neg a \vee c,\ {}_0b,\ {}_0c}{(_1\vee)\ \dfrac{{}_1a,\ {}_1\neg a \vee c^*,\ {}_0b,\ {}_0c}{(_1\neg)\ \dfrac{{}_1a,\ {}_1\neg a^*,\ {}_0b,\ {}_0c}{\dfrac{{}_1a_\times,\ {}_0a_\times,\ {}_0b,\ {}_0c}{\times}}} \qquad \dfrac{{}_1a,\ {}_1c_\times,\ {}_0b,\ {}_0c_\times}{\times}} \qquad \dfrac{{}_1b_\times,\ {}_1\neg a \vee c,\ {}_0b_\times,\ {}_0c}{\times}}$$

The proof tree is closed, therefore the consequence is proven.

**Example 6.6.** Find logical consequences from the following premises: $a$, $a \to b$.

The formulated problem can be considered as a logical equation of the form $a,\ a \to b \models X$ where $X$ is a propositional unknown, that is, its values are propositional formulas; it is required to find solutions to this equation.

We will look for a solution by building a sequent proof tree. We have

$$\frac{{}_1a,\ {}_1a \to b_*,\ {}_0X}{\dfrac{{}_1a,\ {}_0a,\ {}_0X}{\times} \qquad \dfrac{{}_1a,\ {}_1b,\ {}_0X}{}}$$

We have obtained two sequences: ${}_1a,\ {}_0a,\ {}_0X$ and ${}_1a,\ {}_1b,\ {}_0X$. The first sequence is closed, so let's focus on the second sequence. We need to find such a formula that its substitution in $X$ (i.e., replacing $X$ with this formula) gives a true consequence. We proceed from the simple to the complex, namely, we try to close the sequence using the formula ${}_1a$. Then we can take $X$ as ${}_0a$. We get a closed sequence ${}_1a,\ {}_1b,\ {}_0a$. This means that the solution to the original equation has been found.

Let's continue the search for other solutions. Now we try to close the sequent using the formula ${}_1b$. Then we can take $X$ as ${}_0b$. We obtain a closed sequent ${}_1a,\ {}_1b, {}_0b$. This means that the solution of the original equation has been found.

In the sequent ${}_1a,\ {}_1b, {}_0X$, two possibilities of replacing $X$ with formulas that close the sequent have already been considered. The remaining case is when $X$ closes the sequence, which means that $X$ is a tautology.

Thus, three solutions are obtained:

– X is $a$,

– X is $b$,

– X is a tautology.

The first solution is a concretization of the rule that each premise can be considered as a conclusion, the third solution is a concretization of the rule that each tautology does not require any premises (thus, arbitrary premises can be taken), therefore only the second case is relevant to the formulated problem. This case gives an important rule $a,\ a \to b \models b$, which is called modus ponens.

It should also be noted that there are other solutions of the original equation $a,\ a \to b \models X$, which enrich the solutions found, for example, $b \vee B$, where $B$ is an arbitrary formula. We will not consider such solutions here.

**Example 6.7.** Find premises whose logical consequences are the following formula: $a \to a \wedge b$.

We build the equation: $X \models a \to a \wedge b$.

Now we try to build a sequent proof:

$$\dfrac{\dfrac{\dfrac{{}_1X, {}_0a \to a \wedge b}{{}_1X, {}_1a, {}_0a \wedge b}}{\dfrac{{}_1X, {}_1a, {}_0a}{\times}} \quad {}_1X, {}_1a, {}_0b}$$

One branch of the proof is closed, so it remains to analyze the sequent ${}_1X, {}_1a, {}_0b$.

We get three cases:

– $X$ is a contradiction (no branch is closed),

– $X$ is $\neg a$ (because to close the sequence we need ${}_0a$),

– $X$ is $b$ (because to close the sequence we need ${}_1b$).

As noted in the previous example, we do not consider extended solutions.

**Remarks** (application of rules to sequents). So far, we have not described in detail the procedure for applying sequent rules to sequents. It consists of three points:

1) choosing a sequent rule that can be applied to a given (object) sequent;

2) constructing a unifier, that is, such a connection of variables in the bases of the sequent rule with certain propositional formulas (or sequents) that when replacing these variables in the bases of the rule with formulas from the unifier, we obtain an object sequent;

3) applying the constructed unifier to the conclusions of the sequent rule, which gives a conclusion for the object sequent.

For example, we have an object sequent ${}_1A \to (B \to C),\ {}_0(A \to B) \to C$.

Having analyzed it, we apply the rule ${}_0\to$ to the second formula of the sequent. We write the rule in the form

$$\frac{{}_0A \to B, \Sigma}{{}_1A,\ {}_0B, \Sigma}$$

Now we will construct a unifier. The rules' bases specify variables (it's even better to call them metavariables, since their values will be propositional variables or formulas) $A$, $B$, $\Sigma$.

Here $A$ and $B$ are propositional metavariables (since they will be replaced by propositional formulas), and $\Sigma$ is a sequent metavariable (since it will be replaced by a sequent). We single out the formula ${}_0(A \to B) \to C$ that we will transform. We will not transform the

second formula now. Therefore, the sequent metavariable Σ will correspond the formula $_1A \to (B \to C)$ (written as $\Sigma/_1 A \to (B \to C)$), and the formula $_0A \to B$ from the rules' premises will correspond the formula $_0(A \to B) \to C$ from the object sequent. In the object formula, the main operation is the second implication, therefore, during unification, the metavariable *A* from the rule corresponds to the formula (written as $A/A \to B$), and the metavariable *B* to the formula *C* (written as $B/C$). This means that the unifier

$$[\Sigma/_1 A \to (B \to C), A/A \to B, B/C]$$

is constructed.

Now we apply the obtained unifier to the conclusions of the rule

$$_1A, _0B, \Sigma,$$

replacing the metavariables with the corresponding ones from the unifier. We obtain an object conclusion (we put the value Σ at the beginning of the conclusion so that there is a correspondence to the object sequent):

$$_1A \to (B \to C), _1A \to B, _0C.$$

### *6.2. Sequent calculus for classical predicate logic*

Predicate logic, unlike propositional logic, introduces new logical operations – quantifiers. Let us consider sequent rules for quantifiers.

By definition, an indexed formula $_1\exists xA(x)$ is true when interpreted in the domain *M* there is a value of the subject variable *x* that makes the formula *A*(*x*) true. We will denote such a value of *x* by *y*. However, in sequent calculus there is no binding to a specific domain *M*, so we will interpret *y* as a specific subject variable. In other words, the formula $_1\exists xA(x)$ can be replaced by the formula $_1A(y)$, where *y* is the new variable. Taking into account the presence of other formulas of the set in the sequent Σ, we obtain the rule

$$ {}_1\exists\ \frac{{}_1\exists xA(x), \Sigma}{{}_1A(y), \Sigma} $$

provided that the new variable *y* does not occur in $\Sigma \cup \{\exists xA(x)\}$.

Now consider the rule for ${}_0\exists xA(x)$. By definition, this formula is false if the formula $A(x)$ is false for all values of *x* from the domain of subject interpretation. In a sequent, such values are given by its free variables, so it seems that the rule can be written in the form

$$ \frac{{}_0\exists xA(x), \Sigma}{{}_0A(z_1), ..., {}_0A(z_m), \Sigma}, $$

where $\{z_1, \ldots, z_m\}$ are all free variables $\Sigma \cup \{\exists xA(x)\}$.

However, such a rule does not take into account two points:
1) what will happen in the absence of free subject variables in the set of formulas $\Sigma \cup \{\exists xA(x)\}$;
2) will the rule be correct if new free variables appear in the process of derivation?

In the first case, if there are no free variables, then we take an arbitrary new variable, for example $z_1$. In the second case, the situation is more complicated, since the formula $A(x)$ must also be false for the values of the new variables. To take this point into account, we leave the formula ${}_0\exists xA(x)$ in the rule, which, when expanded, will be able to take into account the values of the new variables. Thus, the corrected rule will have the form

$$ {}_0\exists\ \frac{{}_0\exists xA(x), \Sigma}{{}_0A(z_1), ..., {}_0A(z_m), \Sigma, {}_0\exists xA(x)}, $$

where $\{z_1, \ldots, z_m\}$ are all free variables $\Sigma \cup \{\exists xA(x)\}$; if there are no free variables, then we take an arbitrary new variable $z_1$.

Note that in this rule there is no simplification, but new simpler formulas appear, which, perhaps, will make the sequent closed.

Sequent rules for a universal quantifier are constructed dually.

**Remark.** Let us discuss the issue of introducing new variables in the rules ${}_1\exists$ and ${}_0\exists$ of the quantifier elimination. These new variables can be interpreted as constants in the sense that their values will no longer change. Therefore, for such variables, we can use the notations that are used for subject constants: *a*, *b*, *c*, *d*, *e*, …. In other words, the new variables have a dual nature: on the one hand, they are variables, and on the other hand, they can be interpreted as constants. Therefore, rule ${}_1\exists$ can be written as

$$ {}_1\exists \ \frac{{}_1\exists x A(x), \Sigma}{{}_1 A(a), \Sigma} $$

provided that the new variable (constant) *a* does not occur in $\Sigma \cup \{\exists x A(x)\}$, and it can be said that the constant *a* confirms the truth of the predicate *A*.

Similarly, rule ${}_0\exists$ can also be represented by the formula

$$ {}_0\exists \ \frac{{}_0\exists x A(x), \Sigma}{{}_0 A(z_1), \ldots, {}_0 A(z_m), \Sigma, {}_0\exists x A(x)}, $$

where $\{z_1, \ldots, z_m\}$ are all free variables ; if there are no free variables, then we take an arbitrary new variable (constant) $z_1$.

However, such constants in sequent rules are treated as variables, and in interpretations they can be understood as constants (the remark is made for the purpose of additional clarification of the rules).

In sequent rules (forms) for predicate logic, conclusions are written above the dash, premises are written below the dash:

$$ {}_1\neg \ \frac{{}_1\neg A, \Sigma}{{}_0 A, \Sigma} \qquad {}_0\neg \ \frac{{}_0\neg A, \Sigma}{{}_1 A, \Sigma} $$

$$ {}_1\vee \ \frac{{}_1 A \vee B, \Sigma}{{}_1 A, \Sigma \quad {}_1 B, \Sigma} \qquad {}_0\vee \ \frac{{}_0 A \vee B, \Sigma}{{}_0 A, \ {}_0 B, \Sigma} $$

$$ {}_1\wedge \ \frac{{}_1 A \wedge B, \Sigma}{{}_1 A, {}_1 B, \Sigma} \qquad {}_0\wedge \ \frac{{}_0 A \wedge B, \Sigma}{{}_0 A, \Sigma \ \ {}_0 B, \Sigma} $$

$$ {}_1\!\to \frac{{}_1A \to B, \Sigma}{{}_0A, \Sigma \qquad {}_1B, \Sigma} \qquad {}_0\!\to \frac{{}_0A \to B, \Sigma}{{}_1A,\ {}_0B, \Sigma} $$

$$ {}_1\!\leftrightarrow \frac{{}_1A \leftrightarrow B, \Sigma}{{}_0A,\ {}_0B, \Sigma \qquad {}_1A,{}_1B, \Sigma} \qquad {}_0\!\leftrightarrow \frac{{}_0A \leftrightarrow B, \Sigma}{{}_0A,\ {}_1B, \Sigma \qquad {}_1A,{}_0B, \Sigma} $$

$$ {}_1\!\exists\ \frac{{}_1\exists xA(x), \Sigma}{{}_1A(y), \Sigma} $$

provided that the new variable $y$ does not occur in $\Sigma \cup \{\exists xA(x)\}$.

$$ {}_0\!\exists\ \frac{{}_0\exists xA(x), \Sigma}{{}_0A(z_1), ..., {}_0A(z_m), \Sigma, {}_0\exists xA(x)}, $$

where $\{z_1, \ldots, z_m\}$ are all free variables $\Sigma \cup \{\exists xA(x)\}$; if there are no free variables, then an arbitrary new variable $z_1$ is taken.

$$ {}_1\!\forall\ \frac{{}_1\forall xA(x), \Sigma}{{}_1A(z_1), ..., {}_1A(z_m), \Sigma, {}_1\forall xA(x)}, $$

where $\{z_1, \ldots, z_m\}$ are all free variables $\Sigma \cup \{\exists xA(x)\}$; if there are no free variables, then an arbitrary new variable $z_1$ is taken.

$$ {}_0\!\forall\ \frac{{}_0\forall xA(x), \Sigma}{{}_0A(y), \Sigma} $$

provided that the new variable $y$ does not occur in $\Sigma \cup \{\forall xA(x)\}$.

We will assume that all variables are reflected in atomic formulas, the number of which is equal to the arity of the predicate. However, we will not specify free variables in the rule formulas, i.e. the formula $A$ and will not contain a list of all free variables (the same applies to formula $B$ and the set of formulas ).

**Example 6.8.** Prove the formula

$$\forall x(A(x) \to B(x)) \to (\forall xA(x) \to \exists xB(x))$$

We form the marked sequent:

$${}_0\forall x(A(x) \to B(x)) \to (\forall xA(x) \to \exists xB(x)).$$

The main operation is the second implication, therefore we apply the rule ${}_0\to$:

$$({}_0\to)\ \frac{{}_0\forall x(A(x)\to B(x))\to(\forall xA(x)\to\exists xB(x))}{{}_1\forall x(A(x)\to B(x)),\ {}_0\forall xA(x)\to\exists xB(x)}$$

We have obtained a new sequent of two formulas. To the second formula it is advisable to apply the rule ${}_0\to$:

$$({}_0\to)\ \frac{{}_0\forall x(A(x)\to B(x))\to(\forall xA(x)\to\exists xB(x))}{({}_0\to)\ \dfrac{{}_1\forall x(A(x)\to B(x)),\ {}_0\forall xA(x)\to\exists xB(x)*}{{}_1\forall x(A(x)\to B(x)),\ {}_1\forall xA(x),\ {}_0\exists xB(x)}}$$

In the new sequent it is advisable to apply the rule ${}_1\forall$ to the second formula ${}_1\forall xA(x)$. This rule requires us to identify all free variables in the sequent. In this case there are no free variables, therefore we take an arbitrary new variable *y*, which we substitute into the formula $A(x)$ instead of *x*. We have:

$$({}_0\to)\ \frac{{}_0\forall x(A(x)\to B(x))\to(\forall xA(x)\to\exists xB(x))}{({}_0\to)\ \dfrac{{}_1\forall x(A(x)\to B(x)),\ {}_0\forall xA(x)\to\exists xB(x)*}{({}_1\forall)\ \dfrac{{}_1\forall x(A(x)\to B(x)),\ {}_1\forall xA(x)*,\ {}_0\exists xB(x)}{{}_1\forall x(A(x)\to B(x)),\ {}_1A(y),\ {}_1\forall xA(x),\ {}_0\exists xB(x)}}}$$

Furthermore, we can apply the rule ${}_1\forall$ to the first formula, but now we have a free variable *y*. We obtain:

$$({}_0\to)\ \frac{{}_0\forall x(A(x)\to B(x))\to(\forall xA(x)\to\exists xB(x))}{({}_0\to)\ \dfrac{{}_1\forall x(A(x)\to B(x)),\ {}_0\forall xA(x)\to\exists xB(x)*}{({}_1\forall)\ \dfrac{{}_1\forall x(A(x)\to B(x)),\ {}_1\forall xA(x)*,\ {}_0\exists xB(x)}{({}_1\forall)\ \dfrac{{}_1\forall x(A(x)\to B(x))*,\ {}_1A(y),\ {}_1\forall xA(x),\ {}_0\exists xB(x)}{{}_1A(y)\to B(y),\ {}_1\forall x(A(x)\to B(x)),\ {}_1A(y),\ {}_1\forall xA(x),\ {}_0\exists xB(x)}}}}$$

(the last sequent is placed on two lines).

To the last formula of the sequent we apply the rule ${}_0\exists$, taking into account that the only free variable in the sequent is *y*:

$$(_0\to)\ \frac{{}_0\forall x(A(x)\to B(x))\to(\forall xA(x)\to\exists xB(x))}{(_0\to)\ \dfrac{{}_1\forall x(A(x)\to B(x)),\ {}_0\forall xA(x)\to\exists xB(x)*}{(_1\forall)\ \dfrac{{}_1\forall x(A(x)\to B(x)),\ {}_1\forall xA(x)*,\ {}_0\exists xB(x)}{(_1\forall)\ \dfrac{{}_1\forall x(A(x)\to B(x))*,\ {}_1A(y),\ {}_1\forall xA(x),\ {}_0\exists xB(x)}{{}_1A(y)\to B(y),{}_1\ \forall x(A(x)\to B(x)),}}}}$$

$$(_0\exists)\ \frac{{}_1A(y),\ {}_1\forall xA(x),\ {}_0\exists xB(x)*}{{}_1A(y)\to B(y),{}_1\ \forall x(A(x)\to B(x)),\ {}_1A(y),\ {}_1\forall xA(x),B(y),\ {}_0\exists xB(x)}$$

Now we apply the rule ${}_1\to$ to the first formula. Since the obtained sequents are quite large, we will write them in several lines:

$$(_0\to)\ \frac{{}_0\forall x(A(x)\to B(x))\to(\forall xA(x)\to\exists xB(x))}{(_0\to)\ \dfrac{{}_1\forall x(A(x)\to B(x)),\ {}_0\forall xA(x)\to\exists xB(x)*}{(_1\forall)\ \dfrac{{}_1\forall x(A(x)\to B(x)),\ {}_1\forall xA(x)*,\ {}_0\exists xB(x)}{(_1\forall)\ \dfrac{{}_1\forall x(A(x)\to B(x))*,\ {}_1A(y),\ {}_1\forall xA(x),\ {}_0\exists xB(x)}{{}_1A(y)\to B(y),{}_1\ \forall x(A(x)\to B(x)),}}}}$$

$$(_0\exists)\ \frac{{}_1A(y),\ {}_1\forall xA(x),\ {}_0\exists xB(x)*}{{}_1A(y)\to B(y)*,\ {}_1\forall x(A(x)\to B(x)),}$$

$$(_1\to)\ \frac{{}_1A(y),\ {}_1\forall xA(x),\ {}_0B(y),\ {}_0\exists xB(x)}{{}_0A(y)_\times,\ {}_1\forall x(A(x)\to B(x)),\ {}_1A(y)_\times,\ {}_1\forall xA(x),\ {}_0B(y),\ {}_0\exists xB(x)} \qquad \frac{{}_1B(y)_\times,{}_1\ \forall x(A(x)\to B(x)),\ {}_1A(y),\ {}_1\forall xA(x),\ {}_0B(y)_\times,\ {}_0\exists xB(x)}{}$$

$$\times \qquad\qquad \times$$

All obtained sequents are closed, therefore the initial formula is always true (true everywhere, valid, universally true, true for every interpretations).

**Example 6.9.** Is the formula $(\forall xA(x) \to \exists xB(x)) \to \forall x(A(x) \to B(x))$ always true?

In this and subsequent examples, we will not indicate the name of the applied rule, since it is easily determined by the main operation of the formula, indexed by the * sign. We formulate a marked sequent and begin to build a derivation tree:

$$\frac{{}_0(\forall xA(x) \to \exists xB(x)) \to \forall x(A(x) \to B(x))}{\dfrac{{}_1\forall xA(x) \to \exists xB(x)^*,\ {}_0\forall x(A(x) \to B(x))}{{}_0\forall xA(x)^*,\ {}_0\forall x(A(x) \to B(x)) \quad {}_1\exists xB(x)^*,\ {}_0\forall x(A(x) \to B(x))}}$$

We obtained two sequents that do not have free variables. Applying the rule ${}_0\forall$ to the first formula of the first sequent introduces a new variable *y*; when applying the rule ${}_1\exists$ to the first formula of the second sequent, we can take the same new (for the second sequent) variable *y*. We get:

$$\frac{{}_0(\forall xA(x) \to \exists xB(x)) \to \forall x(A(x) \to B(x))}{\dfrac{{}_1\forall xA(x) \to \exists xB(x)^*,\ {}_0\forall x(A(x) \to B(x))}{\dfrac{{}_0\forall xA(x)^*,\ {}_0\forall x(A(x) \to B(x))}{{}_0A(y),\ {}_0\forall x(A(x) \to B(x))} \quad \dfrac{{}_1\exists xB(x)^*,\ {}_0\forall x(A(x) \to B(x))}{{}_1B(y),\ {}_0\forall x(A(x) \to B(x))}}}$$

Both new sequents include a formula ${}_0\forall x(A(x) \to B(x))$, that we transform according to the rule ${}_0\forall$ using a new free variable z. Next, we apply the rule ${}_0\to$. We have:

$$\frac{{}_0(\forall xA(x) \to \exists xB(x)) \to \forall x(A(x) \to B(x))}{\dfrac{{}_1\forall xA(x) \to \exists xB(x)^*,\ {}_0\forall x(A(x) \to B(x))}{\dfrac{{}_0\forall xA(x)^*,\ {}_0\forall x(A(x) \to B(x))}{\dfrac{{}_0A(y),\ {}_0\forall x(A(x) \to B(x))^*}{\dfrac{{}_0A(y),\ {}_0A(z) \to B(z)^*}{\dfrac{{}_0A(y),\ {}_1A(z),\ {}_0B(z)}{!}}}} \quad \dfrac{{}_1\exists xB(x)^*,\ {}_0\forall x(A(x) \to B(x))}{\dfrac{{}_1B(y),\ {}_0\forall x(A(x) \to B(x))^*}{\dfrac{{}_1B(y),\ {}_0A(z) \to B(z)^*}{\dfrac{{}_1B(y),\ {}_1A(z),\ {}_0B(z)}{!}}}}}$$

The obtained sequents ${}_0A(y),\ {}_1A(z),\ {}_0B(z)$ and ${}_1B(y),\ {}_1A(z),\ {}_0B(z)$ contain only atomic formulas, but are not closed. This means that it is possible to construct counterexamples that refute the initial formula. Each of the sequents has two free variables *y* and *z*. Therefore, we choose a model of two constants *a* and *b*, considering them as the values of *y* and *z*, respectively. The first sequent ${}_0A(y),\ {}_1A(z),\ {}_0B(z)$ determines the values of the basic predicates *A* and *B*: $A(a) = 0,\ A(b) = 1, B(b) = 0$ . The sequent does not determine the value $B(a)$, so it can be arbitrary (denoted by the * sign). We construct a table that specifies the model (interpretation of predicate symbols):

| *x* | *A*(*x*) | *B*(*x*) |
|---|---|---|
| *a* | 0 | * |
| *b* | 1 | 0 |

Let's calculate the value of the formula

$$(\forall xA(x) \to \exists xB(x)) \to \forall x(A(x) \to B(x))$$

in this model. Since there are no free variables, the initial state of the variables is given by the empty set. In the given formula, the main operation is implication, so we calculate the values of the antecedent $\forall xA(x) \to \exists xB(x)$ and consequent $\forall x(A(x) \to B(x))$ on the initial state. The antecedent is formed using implication, so we calculate $\forall xA(x)$ and $\exists xB(x)$. We see that in our model $\forall xA(x)$ takes the value 0, respectively, $\exists xB(x)$ gets the value 0 (here the value of the consequent is not important), therefore, the value is 0, that is, the formula is falsifiable.

Similarly, a model is constructed for an unclosed sequent ${}_1B(y),\ {}_1A(z),\ {}_0B(z)$. This model also gives a counterexample. ◄

**Example 6.10.** Is the formula $\exists x(A(x) \to B(x)) \to (A(x) \to \exists xB(x))$ always true?

Note that the formula has a free variable *x*. We construct a sequent derivation of this formula (comments for the derivation rules are written in curly brackets):

$${}_0\exists x(A(x) \to B(x)) \to (A(x) \to \exists xB(x))$$

$${}_1\exists x(A(x) \to B(x)),\ {}_0(A(x) \to \exists xB(x))*$$

$${}_1\exists x(A(x) \to B(x))*,\ {}_1A(x),\ {}_0\exists xB(x)$$

$${}_1A(y) \to B(y))*,\ {}_1A(x),\ {}_0\exists xB(x))$$

| | |
|---|---|
| ${}_0A(y),\ {}_1A(x),\ {}_0\exists xB(x)*$ | ${}_1B(y),\ {}_1A(x),\ {}_0\exists xB(x)*$ |
| {here *x*, *y* are free} | {here *x*, *y* are free} |
| ${}_0A(y),\ {}_1A(x),\ {}_0B(x),\ {}_0B(y),\ {}_0\exists xB(x)$ | ${}_1B(y),\ {}_1A(x),\ {}_0\exists xB(x)*$ |
| {here *x*, *y*, *z* are free} | ${}_1B(y)_\times,\ {}_1A(x),\ {}_0B(y)_\times,\ {}_0B(x),\ {}_0\exists xB(x)$ |
| ! | × |

The left sequent ${}_0A(y),\ {}_1A(x),\ {}_0B(x),\ {}_0B(y),\ {}_0\exists xB(x)$ has a non-atomic formula ${}_0\exists xB(x)$, but applying the rule does not change the sequent. Therefore, the sequent is not closed. To obtain a counterexample, we construct a model with two constants *a* and *b*, considering them to be the values of *x* and *y*, respectively. The sequent determines the values of the basic predicates *A* and *B*: $A(b) = 0,\ A(a) = 1,\ B(a) = 0,\ B(b) = 0$. We build a table that specifies the model (interpretation of predicate symbols):

| *z* | *A*(*z*) | *B*(*z*) |
|---|---|---|
| *a* | 1 | 0 |
| *b* | 0 | 0 |

Let us calculate the value of the formula

$$\exists x(A(x) \to B(x)) \to (A(x) \to \exists x B(x))$$

in such a model, considering $a$ as the value of $x$. In this formula, the main operation is implication, therefore we calculate the values of the antecedent $\exists x(A(x) \to B(x))$ and consequent $(A(x) \to \exists x B(x))$. The main operation of the antecedent is the existential quantifier, therefore we calculate the formula $A(x) \to B(x)$ for the values of $x$ equal to $a$ and $b$. We have that $A(a) \to B(a) = 0$ and $A(b) \to B(b) = 1$. Thus, the value of $\exists x(A(x) \to B(x))$ is 1. Now we calculate the consequent $A(x) \to \exists x B(x)$. Here $A(x)$ takes the value 1, and $\exists x B(x)$ – the value 0. Therefore, $A(x) \to \exists x B(x)$ will have the value 0, as does the entire formula as a whole. Thus, the formula is falsifiable.

Note that if $A$ did not depend on $x$, then the formula $\exists x(A \to B(x)) \to (A \to \exists x B(x))$ would be true. Indeed,

$$\dfrac{{}_0\exists x(A \to B(x)) \to (A \to \exists x B(x))}{\dfrac{{}_1\exists x(A \to B(x)),\ {}_0(A \to \exists x B(x))*}{\dfrac{{}_1\exists x(A \to B(x))*,\ {}_1A,\ {}_0\exists x B(x)}{\dfrac{{}_1A \to B(x)*,\ {}_1A,\ {}_1\exists x B(x)}{\dfrac{{}_0A_{\times},\ {}_1A_{\times},\ {}_0\exists x B(x)}{\times} \qquad \dfrac{{}_1B(x),\ {}_1A,\ {}_0\exists x B(x)*}{\dfrac{{}_1B(x)_{\times},\ {}_1A,\ {}_0B(x)_{\times},\ {}_0\exists x B(x)}{\times}}}}}}$$

**Example 6.11.** Is the formula $\exists x A(x) \vee B(x) \to \exists x(A(x) \wedge B(x))$ always true?

Let's build a sequent derivation:

$$\dfrac{{}_0\exists x A(x) \vee B(x) \to \exists x(A(x) \wedge B(x))}{\dfrac{{}_1\exists x A(x) \vee B(x)*,\ {}_0\exists x(A(x) \wedge B(x))}{{}_1\exists x A(x),\ {}_0\exists x(A(x) \wedge B(x)) \quad {}_1B(x),\ {}_0\exists x(A(x) \wedge B(x))}}$$

Now let's build a derivation for the first sequence:

$$\dfrac{{}_1\exists x A(x)^*,\ {}_0\exists x(A(x)\wedge B(x))}{\dfrac{{}_1A(y),\ {}_0\exists x(A(x)\wedge B(x))^*}{\dfrac{{}_1A(y),\ {}_0A(y)\wedge B(y)^*,\ {}_0\exists x(A(x)\wedge B(x))}{\dfrac{{}_1A(y)_\times,\ {}_0A(y)_\times,\ {}_0\exists x(A(x)\wedge B(x))}{\times}\qquad {}_1A(y),\ {}_0B(y),\ {}_0\exists x(A(x)\wedge B(x))}}}$$

It remains to build a derivation for the second sequence:

$$\dfrac{{}_1A(y),\ {}_0B(y),\ {}_0\exists x(A(x)\wedge B(x))^*}{\dfrac{{}_1A(y),\ {}_0B(y),\ {}_0A(y)\wedge B(y)^*,\ {}_0\exists x(A(x)\wedge B(x))}{\dfrac{{}_1A(y)_\times,\ {}_0B(y),\ {}_0A(y)_\times,\quad {}_1A(y),\ {}_0B(y),\ {}_0B(y),}{{}_0\exists x(A(x)\wedge B(x))\qquad {}_0\exists x(A(x)\wedge B(x))}}}$$

$$\times \qquad\qquad !$$

We have obtained an unclosed sequence, which allows us to build the following model:

| $z$ | $A(z)$ | $B(z)$ |
|---|---|---|
| $a$ | 1 | 0 |

On this model, the initial formula is falsifiable.

**Example 6.12.** Is the formula $\exists y\forall x A(x,y)\to\forall x\exists y A(x,y)$ true everywhere?

Let's build a sequent derivation:

$$\dfrac{{}_0\exists y\forall x A(x,y)\to\forall x\exists y A(x,y)}{\dfrac{{}_1\exists y\forall x A(x,y)^*,\ {}_0\forall x\exists y A(x,y)}{\dfrac{{}_1\forall x A(x,z),\ {}_0\forall x\exists y A(x,y)^*}{\dfrac{{}_1\forall x A(x,z)^*,\ {}_0\exists y A(t,y)}{\dfrac{{}_1\forall x A(x,z),\ {}_1A(z,z),\ {}_1A(t,z),\ {}_0\exists y A(t,y)^*}{\dfrac{{}_1\forall x A(x,z),\ {}_1A(z,z),\ {}_1A(t,z)_\times,\ {}_0A(t,z)_\times,\ {}_0A(t,t),\ {}_0\exists y A(t,y)}{\times}}}}}$$

The derivation tree is closed, so the formula is true everywhere.

**Example 6.13.** Is the formula

$$\forall x \exists y A(x, y) \to \exists y \forall x A(x, y)$$

always true?

We build a sequent derivation (we use the same notation for the introduced variables as for constants: $a$, $b$, $c$, $d$, $e$, …):

$$\frac{{}_0\forall x \exists y A(x, y) \to \exists y \forall x A(x, y)}{{}_1\forall x \exists y A(x, y)^*,\ {}_0\exists y \forall x A(x, y)}$$

{no free variables, introduce $a$}

$$\frac{{}_1\forall x \exists y A(x, y),\ {}_1\exists y A(a, y)^*,\ {}_0\exists y \forall x A(x, y)}{}$$

{introduce new variable $b$}

$$\frac{{}_1\forall x \exists y A(x, y),\ {}_1A(a, b),\ {}_0\exists y \forall x A(x, y)^*}{}$$

{substitute instead of $y$ all free variables $-\ a, b$}

$$\frac{{}_1\forall x \exists y A(x, y),\ {}_1A(a, b),\ {}_0\exists y \forall x A(x, y),\ {}_0\forall x A(x, a)^*,\ {}_0\forall x A(x, b)}{}$$

{introduce new variable $c$}

$$\frac{{}_1\forall x \exists y A(x, y),\ {}_1A(a, b),\ {}_0\exists y \forall x A(x, y),\ {}_0A(c, a),\ {}_0\forall x A(x, b)^*}{}$$

$$\frac{{}_1\forall x \exists y A(x, y)^*,\ {}_1A(a, b),\ {}_0\exists y \forall x A(x, y),\ {}_0A(c, a),\ {}_0A(d, b)}{}$$

$$\frac{{}_1\forall x \exists y A(x, y),\ {}_1\exists y A(b, y)^*,\ {}_1A(a, b),\ {}_0\exists y \forall x A(x, y),\ {}_0A(c, a),\ {}_0A(d, b)}{}$$

{introduce new variable $e$}

$$\frac{{}_1\forall x \exists y A(x, y),\ {}_1A(b, e)^*,\ {}_1A(a, b),\ {}_0\exists y \forall x A(x, y),\ {}_0A(c, a),\ {}_0A(d, b)}{}$$

...

This way we can build an infinite counterexample. In particular, one of the possible falsifiable interpretations is the interpretation on natural numbers, where $A(x, y)$ is interpreted as $x < y$: $\forall x \exists y (x < y) \to \exists y \forall x (x < y)$. This formula is falsifiable.

There is also a finite counterexample, when $A(x, y)$ is given by the table

| $A$ | $a$ | $b$ |
|---|---|---|
| $a$ | 0 | 1 |
| $b$ | 1 | 0 |

From the last two examples it follows that when permuting the quantifiers, the formulas may not be equivalent, that is, the formula $\forall x \exists y A(x, y) \leftrightarrow \exists y \forall x A(x, y)$ is not true everywhere.

**Remark.** There are possible cases when the process of building a proof tree is infinite and it is not possible to build a simple counter-example.

### *6.4. Sequent calculus for pure logic of quasi-ary predicates*

Let us describe the required sequent calculus. The basic condition of the closedness of the sequence ${}_1\Gamma_0\Delta$ with the set of *unv*-variables *Un* is induced by the property C (details can be found in [NikSh2017]):

C) there are *Rs*-*Un*-equivalent formulas $\Psi$ and $\Xi$ such that: $\Psi \in \Gamma$ and $\Xi \in \Delta$; this case includes the classical closedness condition: there exists $\Phi$: $\Phi \in \Gamma$ and $\Phi \in \Delta$.

Basic rules are as follows.

Simplification rules:

${}_1\mathrm{R}\ \dfrac{{}_1\Phi, \Sigma}{{}_{|1}R(\Phi), \Sigma}$; ${}_0\mathrm{R}\ \dfrac{{}_0\Phi, \Sigma}{{}_0R(\Phi), \Sigma}$;

${}_1\mathrm{RI}\ \dfrac{{}_1R^{\bar{v}}_{\bar{x}}(\Phi), \Sigma}{{}_1R^{z,\bar{v}}_{z,\bar{x}}(\Phi), \Sigma}$; ${}_0\mathrm{RI}\ \dfrac{{}_0R^{\bar{v}}_{\bar{x}}(\Phi), \Sigma}{{}_0R^{z,\bar{v}}_{z,\bar{x}}(\Phi), \Sigma}$;

${}_1\mathrm{RU}\ \dfrac{{}_1R^{\bar{v}}_{\bar{u}}(\Phi), \Sigma}{{}_1R^{y,\bar{v}}_{z,\bar{u}}(\Phi), \Sigma}$, if $y \in \nu(\Phi)$; ${}_0\mathrm{RU}\ \dfrac{{}_0R^{\bar{v}}_{\bar{u}}(\Phi), \Sigma}{{}_0R^{y,\bar{v}}_{z,\bar{u}}(\Phi), \Sigma}$, if $y \in \nu(\Phi)$;

${}_1\mathrm{R}\exists\mathrm{R}\ \dfrac{{}_1R^{\bar{u}}_{\bar{v}}(\exists x\Phi), \Sigma}{{}_1R^{\bar{u},x}_{\bar{v},y}(\exists x\Phi), \Sigma}$; ${}_0\mathrm{R}\exists\mathrm{R}\ \dfrac{{}_0R^{\bar{u}}_{\bar{v}}(\exists x\Phi), \Sigma}{{}_0R^{\bar{u},x}_{\bar{v},y}(\exists x\Phi), \Sigma}$.

Rules of equivalent transformations:

$_1\mathrm{RR}\ \dfrac{{}_1R_{\bar{x}}^{\bar{v}} \circ_{\bar{y}}^{\bar{w}}(\Phi), \Sigma}{{}_1R_{\bar{x}}^{\bar{v}}({}_1R_{\bar{y}}^{\bar{w}}(\Phi)), \Sigma}$; $\quad _0\mathrm{RR}\ \dfrac{{}_0R_{\bar{x}}^{\bar{v}} \circ_{\bar{y}}^{\bar{w}}(\Phi), \Sigma}{{}_0R_{\bar{x}}^{\bar{v}}({}_0R_{\bar{y}}^{\bar{w}}(\Phi)), \Sigma}$;

$_1\mathrm{R}\neg\ \dfrac{{}_1R_{\bar{x}}^{\bar{v}}(\Phi), \Sigma}{{}_1R_{\bar{x}}^{\bar{v}}(\neg\Phi), \Sigma}$; $\quad _0\mathrm{R}\neg\ \dfrac{{}_0\neg R_{\bar{x}}^{\bar{v}}(\Phi), \Sigma}{{}_0R_{\bar{x}}^{\bar{v}}(\neg\Phi), \Sigma}$;

$_1\mathrm{R}\vee\ \dfrac{{}_1R_{\bar{x}}^{\bar{v}}(\Phi) \vee R_{\bar{x}}^{\bar{v}}(\Psi), \Sigma}{{}_1R_{\bar{x}}^{\bar{v}}(\Phi \vee \Psi), \Sigma}$; $\quad _0\mathrm{R}\vee\ \dfrac{{}_0R_{\bar{x}}^{\bar{v}}(\Phi) \vee R_{\bar{x}}^{\bar{v}}(\Psi), \Sigma}{{}_0R_{\bar{x}}^{\bar{v}}(\Phi \vee \Psi), \Sigma}$.

Simplifications rules for predicates *E*:

$_1\mathrm{R_E}\ \dfrac{{}_1Ez, \Sigma}{{}_1R_{\bar{x}}^{\bar{v}}(Ez), \Sigma}$, де $z \notin \{\bar{v}\}$; $_0\mathrm{R_E}\ \dfrac{{}_0Ez, \Sigma}{{}_0R_{\bar{x}}^{\bar{v}}(Ez), \Sigma}$, де $z \notin \{\bar{v}\}$;

$_1\mathrm{R_Ev}\ \dfrac{{}_1Ey, \Sigma}{{}_1R_{\bar{x},y}^{\bar{v},z}(Ez), \Sigma}$;

Rules of the decompositions:

$_1\neg\ \dfrac{{}_0\Phi, \Sigma}{{}_1\neg\Phi, \Sigma}$; $\quad _0\neg\ \dfrac{{}_1\Phi, \Sigma}{{}_0\neg\Phi, \Sigma}$;

$_1\vee\ \dfrac{{}_1\Phi, \Sigma \quad {}_1\Psi, \Sigma}{{}_1\Phi \vee \Psi, \Sigma}$; $\quad _0\vee\ \dfrac{{}_0\Phi, {}_0\Psi, \Sigma}{{}_0\Phi \vee \Psi, \Sigma}$.

Rules for quantifier eliminations:

$_1\exists\ \dfrac{{}_1R_z^x(\Phi), {}_1Ez, \Sigma}{{}_1\exists x\Phi, \Sigma}, z \in fu(\Sigma, \exists x\Phi)$; $\quad _0\exists\mathrm{v}$

$\dfrac{{}_0\exists x\Phi, {}_1Ey, {}_0R_y^x(\Phi), \Sigma}{{}_0\exists x\Phi, {}_1Ey, \Sigma}$;

$_1\exists\mathrm{R}\ \dfrac{{}_1R_{\bar{v},z}^{\bar{u},x}(\Phi), {}_1Ez, \Sigma}{{}_1R_{\bar{v}}^{\bar{u}}(\exists x\Phi), \Sigma}, z \in fu(\Sigma, R_{\bar{v}}^{\bar{u}}(\exists x\Phi))$;

$$ {}_0\exists\mathrm{Rv}\ \frac{{}_0R^{\bar{u}}_{\bar{v}}(\exists x\Phi),\ {}_0R^{\bar{u},x}_{\bar{v},y}(\Phi),\ {}_1Ey,\Sigma}{{}_0R^{\bar{u}}_{\bar{v}}(\exists x\Phi),\ {}_1Ey,\Sigma}. $$

For rules ${}_1\exists$R, ${}_1\exists$R ${}_0\exists$v, ${}_0\exists$Rv the formula $Ey$ should not belong to $\Sigma$.

Additional rules for $E$:

Ed $\dfrac{{}_1Ex,\Sigma \quad {}_0Ex,\Sigma}{\Sigma}$ ($Ex$ should not belong to $\Sigma$);

Ev $\dfrac{{}_1Ez,\ \Sigma}{\Sigma}$ if $z\in fu(\Sigma)$.

The constructed calculus is sound and complete.

### *6.5. Sequent calculus for predicate logic over hierarchical nominative data with the complement composition*

Here we treat sequents as finite sets of formulas indexed by symbols ${}_1$, ${}_0$, and ${}_u$. Sequents are denoted ${}_1\Gamma_u U_0\Delta$, in abbreviated form $\Sigma$. Formulas from $\Gamma$ are indexed by ${}_1$, formulas from $\Delta$ are indexed by ${}_0$, and formulas from U are indexed by ${}_u$. Sequent calculi are specified by sequent rules (forms) and closeness conditions. We concentrate on sequent rules. They are syntactical analogs of the semantic properties of the relations of logical irrefutability consequence $|=_{IR}{}^{u}$.

The *sequent rules for compositions* $\vee$, $\neg$, $\sim$ are induced by the corresponding properties of formulas decomposition:

${}_1\neg\ \dfrac{{}_0\Phi,\ \Sigma}{{}_1\neg\Phi,\ \Sigma}$; ${}_0\neg\ \dfrac{{}_1\Phi,\ \Sigma}{{}_0\neg\Phi,\ \Sigma}$; ${}_u\neg\ \dfrac{{}_u\Phi,\ \Sigma}{{}_u\neg\Phi,\ \Sigma}$;

${}_1\vee\ \dfrac{{}_1\Phi,\ \Sigma \quad {}_1\Psi,\ \Sigma}{{}_1\Phi\vee\Psi,\ \Sigma}$; ${}_0\vee\ \dfrac{{}_0\Phi,\ {}_0\Psi,\ \Sigma}{{}_0\Phi\vee\Psi,\ \Sigma}$;

$$ {}_{u}\vee\ \frac{{}_{u}\Phi, {}_{u}\vartheta,\ \Sigma \quad {}_{u}\Phi, {}_{0}\vartheta,\ \Sigma \quad {}_{0}\Phi, {}_{u}\vartheta,\ \Sigma}{{}_{u}\Phi\vee\vartheta,\ \Sigma}; $$

$$ {}_{1}\sim\ \frac{{}_{u}\Phi,\ \Sigma}{{}_{1}\sim\Phi,\ \Sigma}; \qquad {}_{\perp}\sim\ \frac{{}_{1}\Phi,\ \Sigma \quad {}_{0}\Phi,\ \Sigma}{{}_{u}\sim\Phi,\ \Sigma}. $$

Let us explain the rule ${}_{u}\vee$. A consequence of this rule is formula ${}_{u}\Phi\vee\vartheta,\ \Sigma$. According to the definition of disjunction we see that the value of $\Phi\vee\vartheta$ can be undefined in three cases:

- both arguments $\Phi$ and $\vartheta$ are undefined;
- $\Phi$ is undefined and $\vartheta$ is false;
- $\Phi$ is false and $\vartheta$ is undefined.

Therefore, the rule ${}_{u}\vee$ has three premises.

For the *composition of superposition* we use the following sequent rules:

$$ {}_{1}\mathrm{S}\ \frac{{}_{1}\Phi,\Sigma}{{}_{1}S_P(\Phi),\Sigma}; \qquad {}_{0}\mathrm{S}\ \frac{{}_{0}\Phi,\Sigma}{{}_{u}S_P(\Phi),\Sigma}; \qquad {}_{u}\mathrm{S}\ \frac{{}_{u}\Phi,\Sigma}{{}_{u}S_P(\Phi),\Sigma}; $$

$$ {}_{1}\mathrm{SI}\ \frac{{}_{1}S_P^{\bar{u}}(\Phi,\bar{t}),\Sigma}{{}_{1}S_P^{z,\bar{u}}(\Phi,'z,\bar{t}),\Sigma}; \quad {}_{0}\mathrm{SI}\ \frac{{}_{0}S_P^{\bar{u}}(\Phi,\bar{t}),\Sigma}{{}_{0}S_P^{z,\bar{u}}(\Phi,'z,\bar{t}),\Sigma}; \quad {}_{u}\mathrm{SI}\ \frac{{}_{u}S_P^{\bar{u}}(\Phi,\bar{t}),\Sigma}{{}_{u}S_P^{z,\bar{u}}(\Phi,'z,\bar{t}),\Sigma}; $$

$$ {}_{1}\mathrm{SU}\ \frac{{}_{1}S_P^{\bar{v}}(\Phi,\bar{t}),\Sigma}{{}_{1}S_P^{z,\bar{v}}(\Phi,t',\bar{t}),\Sigma}; \quad {}_{0}\mathrm{SU}\ \frac{{}_{0}S_P^{\bar{v}}(\Phi,\bar{t}),\Sigma}{{}_{0}S_P^{z,\bar{v}}(\Phi,t',\bar{t}),\Sigma}; {}_{u}\mathrm{SU}\ \frac{{}_{u}S_P^{\bar{v}}(\Phi,\bar{t}),\Sigma}{{}_{u}S_P^{z,\bar{v}}(\Phi,t',\bar{t}),\Sigma}; $$

$$ {}_{1}\mathrm{SN}\ \frac{{}_{1}\exists z S_P^{x}(\Phi,'z),\Sigma}{{}_{1}\exists x\Phi,\Sigma}; \quad {}_{0}\mathrm{SN}\ \frac{{}_{0}\exists z S_P^{x}(\Phi,'z),\Sigma}{{}_{0}\exists x\Phi,\Sigma}; \quad {}_{u}\mathrm{SN}\ \frac{{}_{u}\exists z S_P^{x}(\Phi,'z),\Sigma}{{}_{u}\exists x\Phi,\Sigma}; $$

$$ {}_{1}\mathrm{S}\neg\ \frac{{}_{1}\neg S_P^{\bar{v}}(\Phi,\bar{t}),\Sigma}{{}_{1}S_P^{\bar{v}}(\neg\Phi,\bar{t}),\Sigma}; \quad {}_{0}\mathrm{S}\neg\ \frac{{}_{0}\neg S_P^{\bar{v}}(\Phi,\bar{t}),\Sigma}{{}_{0}S_P^{\bar{v}}(\neg\Phi,\bar{t}),\Sigma}; \quad {}_{u}\mathrm{S}\neg\ \frac{{}_{u}\neg S_P^{\bar{v}}(\Phi,\bar{t}),\Sigma}{{}_{u}S_P^{\bar{v}}(\neg\Phi,\bar{t}),\Sigma}; $$

$$ {}_{1}\mathrm{S}\vee\ \frac{{}_{1}S_P^{\bar{v}}(\Phi,\bar{t})\vee S_P^{\bar{v}}(\Psi,\bar{t}),\Sigma}{{}_{1}S_P^{\bar{v}}(\Phi\vee\Psi,\bar{t}),\Sigma}; \qquad {}_{0}\mathrm{S}\vee\ \frac{{}_{0}S_P^{\bar{v}}(\Phi,\bar{t})\vee S_P^{\bar{v}}(\Psi,\bar{t}),\Sigma}{{}_{0}S_P^{\bar{v}}(\Phi\vee\Psi,\bar{t}),\Sigma}; $$

$$_{\mathbf{u}}\mathrm{S}\vee\ \frac{{}_{u}S_P^{\bar{v}}(\Phi,\bar{t})\vee S_P^{\bar{v}}(\Psi,\bar{t}),\Sigma}{{}_{u}S_P^{\bar{v}}(\Phi\vee\Psi,\bar{t}),\Sigma};$$

$$_{1}\mathrm{S}\exists\mathrm{s}\ \frac{{}_{1}\exists x(S_P^{\bar{v}}(\Phi,\bar{t})),\Sigma}{{}_{1}S_P^{\bar{v}}(\exists x\Phi,\bar{t}),\Sigma};\ {}_{0}\mathrm{S}\exists\mathrm{s}\quad \frac{{}_{0}\exists x(S_P^{\bar{v}}(\Phi,\bar{t})),\Sigma}{{}_{0}S_P^{\bar{v}}(\exists x\Phi,\bar{t}),\Sigma};\quad {}_{\mathbf{u}}\mathrm{S}\exists\mathrm{s}\ \frac{{}_{u}\exists x(S_P^{\bar{v}}(\Phi,\bar{t})),\Sigma}{{}_{u}S_P^{\bar{v}}(\exists x\Phi,\bar{t}),\Sigma};$$

$$_{1}\mathrm{S}\exists\quad \frac{{}_{1}\exists z\ S_P^{\bar{v},x}(\Phi,\bar{t},'z),\Sigma}{{}_{1}S_P^{\bar{v}}(\exists x\Phi,\bar{t}),\Sigma};\ {}_{0}\mathrm{S}\exists\quad \frac{{}_{0}\exists z\ S_P^{\bar{v},x}(\Phi,\bar{t},'z),\Sigma}{{}_{0}S_P^{\bar{v}}(\exists x\Phi,\bar{t}),\Sigma};\quad {}_{\mathbf{u}}\mathrm{S}\exists\ \frac{{}_{u}\Phi,\Sigma}{{}_{u}S_P^{\bar{v}}(\exists x\Phi,\bar{t}),\Sigma};$$

$$_{1}\mathrm{SE}\ \frac{{}_{1}Ez,\Sigma}{{}_{1}S_P^{\bar{v}}(Ez,\bar{t}),\Sigma};\quad {}_{0}\mathrm{SE}\ \frac{{}_{0}Ez,\Sigma}{{}_{0}S_P^{\bar{v}}(Ez,\bar{t}),\Sigma};\quad {}_{\mathbf{u}}\mathrm{SE}\ \frac{{}_{u}Ez,\Sigma}{{}_{u}S_P^{\bar{v}}(Ez,\bar{t}),\Sigma}.$$

For rules with SU and SN restriction is $z\updownarrow\Phi$; for rules with S$\exists$s restriction is $x\div\bar{v}$; for rules with S$\exists$ restriction is $z\div\bar{v}$, $z\updownarrow\Phi$ and $z\updownarrow\bar{t}$, for rules with SE restriction is $z\div\bar{v}$.

Rules of *quantifier elimination*:

$$_{1}\exists\ \frac{{}_{1}S_P^{x}(\Phi,'z),\ {}_{1}Ez,\Sigma}{{}_{1}\exists x\Phi,\ \Sigma};\quad {}_{0}\exists\ \frac{{}_{0}\exists x\Phi,\ {}_{0}S_P^{x}(\Phi,'y),\ {}_{1}Ey,\ \ \Sigma}{{}_{0}\exists x\Phi,\ {}_{1}Ey,\ \Sigma};$$

$$_{u}\exists\ \frac{{}_{u}S_P^{x}(\Phi,'z),\ {}_{u}\exists x\Phi,\ {}_{1}Ez,\ \Sigma}{{}_{u}\exists x\Phi,\ \Sigma}.$$

For rules ${}_{1}\exists$ and ${}_{u}\exists$ name $z$ should be a fresh unessential name.

Forms for *insertion of variable assignment predicates*:

ud $\dfrac{{}_1Ey,\Sigma \quad {}_0Ey,\Sigma}{\Sigma}$ $({}_1Ey \notin \Sigma)$; uv $\dfrac{{}_1Ez,\ \Sigma}{\Sigma}$ ($z$ is a fresh unessential name).

**Theorem (soundness).** Sequent rules are sound with respect to the consequence relation $|=_{IR}^{u}$.

**Proof** is based on properties of consequence relation presented in this section and in [NikMam2019].

The consequence relations $|=_{IR}^{u}$ can be used to construct derived consequence relations with increasing truth domains $|=_{IR,T}^{u}$ or decreasing falsity domains $|=_{IR,F}^{u}$. Such relations are used in the consequent rule of program logic.

### *6.6. Floyd–Hoare calculus for programs with partial predicates*

Here we present a calculus of Floyd-Hoare type for program logic with partial predicates. Traditional Floyd-Hoare systems are oriented on total pre- and post-conditions, and in the case of partial conditions (predicates) they become unsound. To cope with this difficulty, we extend program logics with the composition of predicate complement. This permit to modify rules of Floyd-Hoare logic in such a way that they become sound. The negative side of this proposal is that logic becomes more complicated because undefinedness conditions for predicates should be taken into account.

The rules are as follows $(P,Q,R_1,R_2,P',Q',b \in Fr^S,\ t \in Fn^S,\ t \in Tr^S,\ S,S_1,S_2 \in Pt^S,\ x \in V^+)$:

AS) $\{S_P^x(P,t)\}AS^x(t)\{P\}$

ID) $\{P\}\ id\ \underline{\{P\}}$

SEQ) $\dfrac{\{P\}\ S_1\ \{Q\},\{Q\}\ S_2\ \{R_1\},\{\sim Q\}\ S_2\ \{R_2\}}{\{P\}\ S_1 \bullet S_2\ \{R_1 \vee R_2\}}$

IF) $\dfrac{\{b \wedge P\}\ S_1\ \{Q\}, \{\neg b \wedge P\}\ S_2\ \{Q\}}{\{P\}\ IF(\ b, S_1, S_2\ \{Q\}}$

WH) $\dfrac{\{b \wedge P\}\ S\ \{P\}, \{b \wedge (\sim P)\}\ S\ \{P\}}{\{P\}\ WH(b,\ S)\ \{\neg b \wedge P\}}$

C) $\dfrac{\{P'\}\ S\ \{Q'\}}{\{P\}\ S\ \{Q\}}$, $P \models_{IR,T}^{\perp} P'$, $Q' \models_{IR,T}^{\perp} Q$

We can prove that the constructed calculus is sound.

For further applications of the presented system, let us construct extended rule for IF-composition.

Let $\vdash\{P_1\}\ S1\{Q\}, \vdash\{P_2\}\ S2\{Q\}$. Note that both triples have the same post-condition but different pre-conditions.

By C rule we can prove

$$\frac{\{P_1\}\ S_1\ \{Q\}}{\{b \wedge P_1\}\ S_1\ \{Q\}} \text{ because } b \wedge P_1 \to P_1$$

and

$$\frac{\{P_2\}\ S_2\ \{Q\}}{\{\neg b \wedge P_2\}\ S_2\ \{Q\}} \text{ because } \neg b \wedge P_2 \to P_2$$

Then:

$$\frac{\{b \wedge P_1\}\ S_1\ \{Q\}}{\{b \wedge (b \wedge P_1 \vee \neg b \wedge P_2)\}\ S_1\ \{Q\}}$$

because $b \wedge (b \wedge P_1 \vee \neg b \wedge P_2) \to b \wedge P_1$

Indeed, using sequent calculus we obtain

$$\dfrac{\dfrac{\dfrac{\dfrac{{}_0 b \wedge (b \wedge P_1 \vee \neg b \wedge P_2) \to b \wedge P_1}{{}_1 b \wedge (b \wedge P_1 \vee \neg b \wedge P_2), {}_0 b \wedge P_1}}{{}_1 b, {}_1 (b \wedge P_1 \vee \neg b \wedge P_2), {}_0 b \wedge P_1}}{\dfrac{{}_1 b, {}_1 (b \wedge P_1 \vee \neg b \wedge P_2), {}_0 b}{\times} \qquad \dfrac{{}_1 b, {}_1 (b \wedge P_1 \vee \neg b \wedge P_2), {}_0 P_1}{\dfrac{{}_1 b, {}_1 (b \wedge P_1), {}_0 P_1}{\dfrac{{}_1 b, {}_1 b, {}_1 P_1, {}_0 P_1}{\times}} \qquad \dfrac{{}_1 b, {}_1 (\neg b \wedge P_2), {}_0 P_1}{\dfrac{{}_1 b, {}_1 \neg b, {}_1 P_2, {}_0 P_1}{\dfrac{{}_1 b, {}_0 b, {}_1 P_2, {}_0 P_1}{\times}}}}}}{\times}$$

In the same way we can prove that

$$\dfrac{\{\neg b \wedge P_2\}\, S_2\, \{Q\}}{\{\neg b \wedge (b \wedge P_1 \vee \neg b \wedge P_2)\}\, S_2\, \{Q\}}$$

because $\neg b \wedge (b \wedge P_1 \vee \neg b \wedge P_2) \to \neg b \wedge P_2$

Indeed

$$\dfrac{\dfrac{\dfrac{\dfrac{{}_0 \neg b \wedge (b \wedge P_1 \vee \neg b \wedge P_2) \to \neg b \wedge P_2}{{}_1 \neg b \wedge (b \wedge P_1 \vee \neg b \wedge P_2), {}_0 \neg b \wedge P_2}}{{}_1 \neg b, {}_1 (b \wedge P_1 \vee \neg b \wedge P_2), {}_0 \neg b \wedge P_2}}{\dfrac{{}_1 \neg b, {}_1 (b \wedge P_1 \vee \neg b \wedge P_2), {}_0 \neg b}{\times} \qquad \dfrac{{}_1 \neg b, {}_1 (b \wedge P_1 \vee \neg b \wedge P_2), {}_0 P_2}{\dfrac{{}_1 \neg b, {}_1 (b \wedge P_1), {}_0 P_2}{\dfrac{{}_1 \neg b, {}_1 b, {}_1 P_1, {}_0 P_2}{\dfrac{{}_0 b, {}_1 b, {}_1 P_1, {}_0 P_2}{\times}}}} \qquad \dfrac{{}_1 \neg b, {}_1 (\neg b \wedge P_2), {}_0 P_2}{\dfrac{{}_1 \neg b, {}_1 \neg b, {}_1 P_2, {}_0 P_2}{\times}}}}{\times}$$

Now we can apply rule *IF* :

$$\dfrac{\{b \wedge (b \wedge P_1 \vee \neg b \wedge P_2)\}\, S1\, \{Q\}, \{\neg b \wedge (b \wedge P_1 \vee \neg b \wedge P_2)\}\, S2\, \{Q\}}{\{(b \wedge P_1 \vee \neg b \wedge P_2)\}\ \textbf{if}\ b\ \textbf{then}\ S1\ \textbf{else}\ S2\, \{Q\}}$$

We obtain the derived rule for IF:

$$\frac{\{P_1\}\, S1\,\{Q\}, \{P_2\}\, S2\,\{Q\}}{\{(b \wedge P_1 \vee \neg b \wedge P_2)\}\ \textbf{if}\ b\ \textbf{then}\ S1\ \textbf{else}\ S2\,\{Q\}}\ \mathit{IFext}$$

**Note:** There are two languages of a program logic:

- a language based on program algebra terms;
- a language using the syntax of traditional programming languages.

The first language is often called *abstract syntax*. The second is the language of *concrete syntax*.

Now we will present the language of calculus based on concrete syntax.

Extended table of Hoare rules (concrete syntax):

| Rule | Name of the rule |
|---|---|
| $\{P[x \mapsto a]\}\ x := a\ \{P\}$ | *AS* |
| $\{P\}\ skip\ \{P\}$ | *skip* |
| $\dfrac{\{P\}\ S1\ \{Q\}, \{Q\}\ S2\ \{R\}}{\{P\}\ S1; S2\ \{R\}}$ | *S* |
| $\dfrac{\{P\}\ S1\ \{Q\}, \{P\}\ S2\ \{Q\}}{\{P\}\ S1 \cup S2\ \{Q\}}$ | *NC* |
| $\dfrac{\{b \wedge P\}\ S1\ \{Q\}, \{\neg b \wedge P\}\ S2\ \{Q\}}{\{P\}\ \textbf{if}\ b\ \textbf{then}\ S1\ \textbf{else}\ S2\ \{Q\}}$ | *IF* |
| $\dfrac{\{P_1\}\ S1\ \{Q\}, \{P_2\}\ S2\ \{Q\}}{\{(b \wedge P_1 \vee \neg b \wedge P_2)\}\ \textbf{if}\ b\ \textbf{then}\ S1\ \textbf{else}\ S2\ \{Q\}}$ | *IFext* |
| $\dfrac{\{b \wedge P\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ S\ \{\neg b \wedge P\}}$ | *WH* |
| $\dfrac{\{P'\}\ S\ \{Q'\}}{\{P\}\ S\ \{Q\}}$, if $P \Rightarrow P'$, $Q' \Rightarrow Q$ | *C* |
| $\dfrac{\{P\}\ S\ \{Q\}}{\{P\}\ \textbf{begin}\ S\ \textbf{end}\ \{Q\}}$ | *BE* |

**Example 6.14.** *Prove or refute the following Hoare triple*:

$$\{(x > 0 \wedge y > 1\}\ \textit{if}\ x + y > 10\ \textit{then}\ x := x - y\ \textit{else}\ y := 2 * x - y\ \{x + y > 10 \wedge x > 4\}$$

*Solution.* To prove this triple, we should construct Hoare triples for assignment operators in the backward mode. We start with the first assignment:

$$AS: \{(x + y > 10 \wedge x > 4)[x \mapsto x - y]\} x := x - y\ \{x + y > 10 \wedge x > 4\}$$

After substitution we obtain:

$$\{(x-y+y>10 \wedge x-y>4)\}x := x-y\ \{x+y>10 \wedge x>4\} \text{ and}$$

$$\{(x>10 \wedge x-y>4)\}x := x-y\ \{x+y>10 \wedge x>4\}$$

For the assignment in the else-part we obtain:

$$\{(x+y>10 \wedge x>4)[y \mapsto 2*x-y]\}y := 2*x-y\ \{x+y>10 \wedge x>4\}$$

After substitution we get:

$$\{(x+2*x-y>10 \wedge x>4)\}y := 2*x-y\ \{x+y>10 \wedge x>4\}$$

Hence,

$$\{(3*x-y>10 \wedge x>4)\}y := 2*x-y\ \{x+y>10 \wedge x>4\}$$

What to do next?

We can take a conjunction of the obtained preconditions:

$$(x>10 \wedge x-y>4) \wedge (3*x-y>10 \wedge x>4)$$

This will be a correct derivation for *if*-statement but the obtained condition is too strong and we can have incorrect refuting data.

Therefore, we take the following precondition for both sides of *if*-statement (*IFext*-rule):

$$(x+y>10) \wedge (x>10 \wedge x-y>4) \vee (\neg(x+y>10) \wedge (3*x-y>10 \wedge x>4))$$

Using *IFext*-rule we obtain:

$$\{(x+y>10) \wedge (x>10 \wedge x-y>4) \vee (\neg(x+y>10) \wedge (3*x-y>10 \wedge x>4))\}$$

$$\textit{if } x+y>10 \textit{ then } x := x-y \textit{ else } y := 2*x-y$$

$$\{x+y>10 \wedge x>4\}$$

To prove the initial Hoare triple we should prove the following implication:

$$(x>0 \wedge y>1) \rightarrow (x+y>10) \wedge (x>10 \wedge x-y>4)$$

$$\vee(\neg(x+y>10) \wedge (3*x-y>10 \wedge x>4))$$

We use the sequent calculus:

$$
\begin{array}{l}
{}_0(x>0\wedge y>1)\to(x+y>10)\wedge(x>10\wedge x-y>4)\\
\vee(\neg(x+y>10)\wedge(3*x-y>10\wedge x>4))\\
\hline
{}_1(x>0\wedge y>1),\ {}_0(x+y>10)\wedge(x>10\wedge x-y>4)\\
\vee(\neg(x+y>10)\wedge(3*x-y>10\wedge x>4))\\
\hline
{}_1(x>0\wedge y>1),\ {}_0(x+y>10)\wedge(x>10\wedge x-y>4),\\
{}_0(\neg(x+y>10)\wedge(3*x-y>10\wedge x>4))\\
\hline
{}_1(x>0),\ {}_1(y>1),\ {}_0(x+y>10)\wedge(x>10\wedge x-y>4),\\
{}_0((x+y\le 10)\wedge(3*x-y>10\wedge x>4))\\
\hline
{}_1(x>0),\ {}_1(y>1),\ {}_1\neg(x+y>10)\wedge(x>10\wedge x-y>4),\\
{}_1\neg((x+y\le 10)\wedge(3*x-y>10\wedge x>4))\\
\hline
{}_1(x>0),\ {}_1(y>1),\ {}_1(x+y\le 10)\vee(x\le 10\vee x-y\le 4),\\
{}_1((x+y>10)\vee(3*x-y\le 10\vee x\le 4))\\
\hline
{}_1(x>0),\ {}_1(y>1),\ {}_1((x+y\le 10)\vee(x\le 10)\vee(x-y\le 4)),\\
{}_1((x+y>10)\vee(3*x-y\le 10)\vee(x\le 4))
\end{array}
$$

From the last sequent (by unfolding the disjunctions), we get several new sequents:

$$
\begin{array}{l}
{}_1(x>0),\ {}_1(y>1),\ {}_1(x+y\le 10),{}_1(x+y>10)\\
{}_1(x>0),\ {}_1(y>1),\ {}_1(x+y\le 10),{}_1(3*x-y\le 10)\\
{}_1(x>0),\ {}_1(y>1),\ {}_1(x+y\le 10),{}_1(x\le 4)\\
{}_1(x>0),\ {}_1(y>1),\ {}_1(x\le 10\vee x-y\le 4),{}_1((x+y>10)\vee(3*x-y\le 10\vee x\le 4))\\
{}_1(x>0),\ {}_1(y>1),\ {}_1(x+y\le 10)\vee(x\le 10\vee x-y\le 4),\\
{}_1((x+y>10)\vee(3*x-y\le 10\vee x\le 4))
\end{array}
$$

After further transformations we get several sequents. Among them:

$$
{}_1(x>0),\ {}_1(y>1),\ {}_1(x+y\le 10)
$$

This gives a system of inequalities:

$$
\begin{cases}
x>0\\
y>1\\
x\le 10
\end{cases}
$$

This system has solutions, for example $x = 2, y = 3$.

This solution refutes the given Hoare triple. Indeed, the precondition is true on this state; after executing of the program we obtain $x = 2, y = 1$; and the post-condition is false on this state.

Thus, the given Hoare-triple is not valid.

**Example 6.15.** *Prove or refute the following Hoare triple*:

$$\{x > 0 \wedge y > 1\} \textit{ if } x + y > 10 \textit{ then } x := x + 2y \textit{ else } y := 2x + y \ \{ x + y > 10 \wedge x > 4\}$$

*Solution.* To prove this triple, we should construct Hoare triples for assignment operators in the backward mode. We start with the first assignment:

$$\{ (x + y > 10 \wedge x > 4)[x \mapsto x + 2y]\} \ x := x + 2y \ \{ x + y > 10 \wedge x > 4\}$$

After substitution we obtain:

$$\{ (x + 2y + y > 10 \wedge x + 2y > 4)\} \ x := x + 2y \ \{ x + y > 10 \wedge x > 4\}$$

then

$$\{ (x + 3y > 10 \wedge x + 2y > 4)\} \ x := x + 2y \ \{ x + y > 10 \wedge x > 4\}$$

For the assignment in the else–part we obtain:

$$\{(x + y > 10 \wedge x > 4)[y \mapsto 2x + y]\} \ y := 2x + y \ \{ x + y > 10 \wedge x > 4\}$$

After substitution we get:

$$\{(x + 2x + y > 10 \wedge x > 4)\} \ y := 2x + y \ \{ x + y > 10 \wedge x > 4\}$$

Hence,

$$\{(3x + y > 10 \wedge x > 4)\} \ y := 2x + y \ \{ x + y > 10 \wedge x > 4\}$$

Using *IFext*-rule we obtain:

$$\{(x + 3y > 10 \wedge x + 2y > 4)\} \ x := x + 2y \ \{ x + y > 10 \wedge x > 4\}$$

$$\{(3x + y > 10 \wedge x > 4)\} \ y := 2x + y \ \{ x + y > 10 \wedge x > 4\}$$

---

$$\{(x + y > 10) \wedge (x + 3y > 10 \wedge x + 2y > 4) \vee \neg(x + y > 10) \wedge (3x + y > 10 \wedge x > 4)\}$$

$$\textit{if } x + y > 10 \textit{ then } x := x + 2y \textit{ else } y := 2x + y$$

$$\{ x + y > 10 \wedge x > 4\}$$

To prove the initial Hoare triple we should prove the following implication:

$$(x>0\wedge y>1)\to(x+y>10)\wedge(x+3y>10\wedge x+2y>4)$$
$$\vee\neg(x+y>10)\wedge(3x+y>10\wedge x>4)$$

We use sequent calculus:

$$\frac{\begin{array}{l}{}_0(x>0\wedge y>1)\to(x+y>10)\wedge(x+3y>10\wedge x+2y>4)\\ \vee\neg(x+y>10)\wedge(3x+y>10\wedge x>4)\end{array}}{\begin{array}{l}{}_1(x>0\wedge y>1),\ {}_0(x+y>10)\wedge(x+3y>10\wedge x+2y>4)\\ \vee\neg(x+y>10)\wedge(3x+y>10\wedge x>4)\end{array}}$$

$$\frac{\begin{array}{l}{}_1(x>0),\ {}_1y>1),\ {}_0(x+y>10)\wedge(x+3y>10\wedge x+2y>4)\\ \vee\neg(x+y>10)\wedge(3x+y>10\wedge x>4)\end{array}}{\begin{array}{l}{}_1x>0,\ {}_1y>1,\ {}_0(x+y>10)\wedge(x+3y>10\wedge x+2y>4),\\ {}_0\neg(x+y>10)\wedge(3x+y>10\wedge x>4)\end{array}}$$

$$\frac{{}_1x>0,\ {}_1y>1,\ {}_0(x+y>10)\wedge(x+3y>10\wedge x+2y>4),{}_0\ \neg(x+y>10)}{{}_1x>0,\ {}_1y>1,\ {}_0(x+y>10)\wedge(x+3y>10\wedge x+2y>4),{}_0\ (3x+y>10\wedge x>4)}$$

$$\frac{{}_1x>0,\ {}_1y>1,\ {}_0(x+y>10),{}_0\ \neg(x+y>10)}{\begin{array}{l}{}_1x>0,\ {}_1y>1,\ {}_0(x+3y>10\wedge x+2y>4),{}_0\ \neg(x+y>10)\\ {}_1x>0,\ {}_1y>1,\ {}_0(x+y>10)\wedge(x+3y>10\wedge x+2y>4),{}_0\ (3x+y>10)\\ {}_1x>0,\ {}_1y>1,\ {}_0(x+y>10)\wedge(x+3y>10\wedge x+2y>4),{}_0\ (x>4)\end{array}}$$

Four sequents were obtained:

${}_1x>0,\ {}_1y>1,\ {}_0(x+y>10),{}_0\ \neg(x+y>10)$

${}_1x>0,\ {}_1y>1,\ {}_0(x+3y>10\wedge x+2y>4),{}_0\ \neg(x+y>10)$

${}_1x>0,\ {}_1y>1,\ {}_0(x+y>10)\wedge(x+3y>10\wedge x+2y>4),{}_0\ (3x+y>10)$

${}_1x>0,\ {}_1y>1,\ {}_0(x+y>10)\wedge(x+3y>10\wedge x+2y>4),{}_0\ (x>4)$

Continue with each sequent separately. The first sequent is closed:

$$\frac{{}_1x>0,\ {}_1y>1,\ {}_0(x+y>10),{}_0\ \neg(x+y>10)}{{}_1x>0,\ {}_1y>1,\ {}_0(x+y>10),{}_1\ (x+y>10)}$$
$$\times$$

Continue with second sequent:

$$\frac{{}_1x>0,\ {}_1y>1,\ {}_0(x+3y>10\wedge x+2y>4),_0\neg(x+y>10)}{\dfrac{{}_1x>0,\ {}_1y>1,\ {}_0(x+3y>10\wedge x+2y>4),_1(x+y>10)}{\dfrac{{}_1x>0,\ {}_1y>1,\ {}_0(x+3y>10),_1(x+y>10)}{{}_1x>0,\ {}_1y>1,\ {}_1(x+3y\le 10),_1(x+y>10)}} \quad \dfrac{{}_1x>0,\ {}_1y>1,\ {}_0(x+2y>4),_1(x+y>10)}{{}_1x>0,\ {}_1y>1,\ {}_1(x+2y\le 4),_1(x+y>10)}}$$

Sequent ${}_1x>0,\ {}_1y>1,\ {}_1(x+3y\le 10),_1(x+y>10)$ gives the system of inequalities:

$$\begin{cases} x>0, \\ y>1, \\ x+3y\le 10, \\ x+y>10. \end{cases}$$

This system does not have a solution.

Let us take sequent ${}_1x>0,\ {}_1y>1,\ {}_1(x+2y\le 4),_1(x+y>10)$. It gives the system of inequalities:

$$\begin{cases} x>0, \\ y>1, \\ x+2y\le 4, \\ x+y>10. \end{cases}$$

This system does not have a solution.

Let us go to the third sequent:

$$\frac{{}_1x>0,\ {}_1y>1,\ {}_0(x+y>10)\wedge(x+3y>10\wedge x+2y>4),\ {}_0(3x+y>10)}{\dfrac{{}_1x>0,\ {}_1y>1,\ {}_0(x+y>10),_0(3x+y>10) \quad {}_1x>0,\ {}_1y>1,\ {}_0(x+3y>10\wedge x+2y>4),_0(3x+y>10)}{{}_1x>0,\ {}_1y>1,\ {}_1x+y\le 10,_1 3x+y\le 10 \quad {}_1x>0,\ {}_1y>1,\ {}_0(x+3y>10\wedge x+2y>4),_0(3x+y>10)}}$$

Sequent ${}_1x > 0,\ {}_1y > 1,\ {}_1x + y \le 10,\ {}_1 3x + y \le 10$ generates the system:

$$\begin{cases} x > 0, \\ y > 1, \\ x + y \le 10, \\ 3x + y \le 10. \end{cases}$$

This system has solutions, say, $x = 2, y = 2$. It refutes the initial Hoare triple.

**Example 6.16.** *Prove or refute the following Hoare triple*:

$$\{(x > 0 \wedge y > 1\}$$
$$\textit{if}\ \ 2x + y > 10\ \textit{then}\ x := 3x + y\ \textit{else}\ \ x := 4x + 3y$$
$$\{\ x + y > 10 \wedge x > 4\}$$

*Solution.* To prove this triple, we should construct Hoare triples for assignment operators in the backward mode. We start with the first assignment:

$$\{\ (x + y > 10 \wedge x > 4)[x \mapsto 3x + y]\}\ x := 3x + y\ \ \{\ x + y > 10 \wedge x > 4\}$$

After substitution we obtain:

$$\{\ (3x + y + y > 10 \wedge 3x + y > 4)\}\ x := 3x + y\ \ \{\ x + y > 10 \wedge x > 4\}$$

then

$$\{\ (3x + 2y > 10 \wedge 3x + y > 4)\}\ x := 3x + y\ \ \{\ x + y > 10 \wedge x > 4\}$$

For the assignment in the else-part we obtain:

$$\{\ (x + y > 10 \wedge x > 4)[x \mapsto 4x + 3y]\}\ x := 4x + 3y\ \ \{\ x + y > 10 \wedge x > 4\}$$

After substitution we obtain:

$$\{\ (4x + 3y + y > 10 \wedge 4x + 3y > 4)\}\ x := 4x + 3y\ \ \{\ x + y > 10 \wedge x > 4\}$$

then

$$\{\ (4x + 4y > 10 \wedge 4x + 3y > 4)\}\ x := 4x + 3y\ \ \{\ x + y > 10 \wedge x > 4\}$$

Using *IFext*-rule we obtain:

$$\frac{\{\,(3x+2y>10\wedge 3x+y>4)\}\ x:=3x+y\ \ \{\,x+y>10\wedge x>4\}\quad \{\,(4x+4y>10\wedge 4x+3y>4)\}\ x:=4x+3y\ \ \{\,x+y>10\wedge x>4\}}{\begin{array}{l}\{(2x+y>10)\wedge(3x+2y>10\wedge 3x+y>4)\\ \vee\neg(2x+y>10)\wedge(4x+4y>10\wedge 4x+3y>4)\}\\ \textit{if}\ \ 2x+y>10\ \textit{then}\ x:=3x+y\ \textit{else}\ \ x:=4x+3y\\ \{\,x+y>10\wedge x>4\}\end{array}}$$

To prove the initial Hoare triple we should prove the following implication:

$$(x>0\wedge y>1)\to(2x+y>10)\wedge(3x+2y>10\wedge 3x+y>4)\\ \vee\neg(2x+y>10)\wedge(4x+4y>10\wedge 4x+3y>4)$$

We use sequent calculus:

$$\frac{{}_0(x>0\wedge y>1)\to(2x+y>10)\wedge(3x+2y>10\wedge 3x+y>4)\\ \vee\neg(2x+y>10)\wedge(4x+4y>10\wedge 4x+3y>4)}{\begin{array}{l}{}_1(x>0\wedge y>1),\ {}_0(2x+y>10)\wedge(3x+2y>10\wedge 3x+y>4)\\ \vee\neg(2x+y>10)\wedge(4x+4y>10\wedge 4x+3y>4)\end{array}}$$

$$\frac{\begin{array}{l}{}_1x>0,\ {}_1y>1,\ {}_0(2x+y>10)\wedge(3x+2y>10\wedge 3x+y>4),\\ {}_0\neg(2x+y>10)\wedge(4x+4y>10\wedge 4x+3y>4)\end{array}}{\begin{array}{l}{}_1x>0,\ {}_1y>1,\ {}_0(2x+y>10),\ {}_0\neg(2x+y>10)\\ {}_1x>0,\ {}_1y>1,\ {}_0(2x+y>10),\ {}_0(4x+4y>10)\\ {}_1x>0,\ {}_1y>1,\ {}_0(2x+y>10),\ {}_0(4x+3y>4)\\ {}_1x>0,\ {}_1y>1,\ {}_0((3x+2y>10\wedge 3x+y>4),\ {}_0\neg(2x+y>10)\\ {}_1x>0,\ {}_1y>1,\ {}_0((3x+2y>10\wedge 3x+y>4),\ {}_0(4x+4y>10)\\ {}_1x>0,\ {}_1y>1,\ {}_0((3x+2y>10\wedge 3x+y>4),\ {}_0(4x+3y>4)\end{array}}$$

The first three sequents generate the following systems:

$$\begin{cases}x>0,\\ y>1,\\ 2x+y\le 10,\\ 2x+y>10.\end{cases}\qquad\begin{cases}x>0,\\ y>1,\\ 2x+y\le 10,\\ 4x+4y\le 10.\end{cases}\qquad\begin{cases}x>0,\\ y>1,\\ 2x+y\le 10,\\ 4x+3y\le 4.\end{cases}$$

The first system is obviously contradictory, in the second and third systems a contradiction arise between equations 1, 2, 4.

Let us continue with the last three sequents:

$${}_1x > 0,\ {}_1y > 1,\ {}_0((3x + 2y > 10 \wedge 3x + y > 4),\ {}_0\neg\,(2x + y > 10)$$

$${}_1x > 0,\ {}_1y > 1,\ {}_0((3x + 2y > 10 \wedge 3x + y > 4),\ {}_0(4x + 4y > 10)$$

$${}_1x > 0,\ {}_1y > 1,\ {}_0((3x + 2y > 10 \wedge 3x + y > 4),\ {}_0(4x + 3y > 4)$$

They generate the following six systems:

$$\begin{cases} x > 0, \\ y > 1, \\ 3x + 2y \le 10, \\ 2x + y > 10. \end{cases} \quad \begin{cases} x > 0, \\ y > 1, \\ 3x + y \le 4, \\ 2x + y > 10. \end{cases} \quad \begin{cases} x > 0, \\ y > 1, \\ 3x + 2y \le 10, \\ 4x + 4y \le 10. \end{cases}$$

$$\begin{cases} x > 0, \\ y > 1, \\ 3x + y \le 4, \\ 4x + 4y \le 10. \end{cases} \quad \begin{cases} x > 0, \\ y > 1, \\ 3x + 2y \le 10, \\ 4x + 3y \le 4. \end{cases} \quad \begin{cases} x > 0, \\ y > 1, \\ 3x + y \le 4, \\ 4x + 3y \le 4. \end{cases}$$

They all are contradictory.

Thus, the initial Hoare triple is valid.

**Example 6.17.** Prove in Floyd–Hoare logic the partial correctness of the GCD program for calculating the greatest common divisor using the Euclidean algorithm.

*Solution.* The GCD program:

*GCD* ≡

**begin**

**while** ¬ *M* = *N* **do**

**if** *M* > *N* **then** *M* := *M* – *N* **else** *N* := *N* – *M*

**end**

First, we formulate an assertion that specifies the correctness of the GCD program. To do this, we introduce the binary function *gcd*: $Int^2 \to Int$,, which specifies the greatest common divisor of its two arguments. Therefore, the notation *gcd* (*m*, *n*) specifies the greatest common divisor of the numbers *m* and *n*. We assume that the initial state of the program has the following form: [$M \mapsto m$, $N \mapsto n$]. This means that we consider a state with variables *M* and *N* and parameters *m* and *n*. Our assumption can be written as a predicate $M > 0 \wedge N > 0 \wedge M = m \wedge N = n$, which will be the precondition of the required assertion. Then the postcondition can be written as $M = N \wedge M = gcd(m,n)$, and it states that after the program is completed, the values of the variables *M* and *N* are equal and equal to the value of *gcd* (*m*, *n*).

So, we need to prove the following assertion, which we will write in three lines:

$$\{M > 0 \wedge N > 0 \wedge M = m \wedge N = n\}$$
$$GCD$$
$$\{M = N \wedge M = gcd(m, n)\}.$$

The idea of constructing the proof is to find the invariant of the cycle. The preliminary consideration of the *GCD* program shows that subtracting a smaller number from a larger one does not change the greatest common divisor. Therefore, we can take the predicate $gcd(m, n) = gcd(M, N)$, which we will denote by *P*, as the invariant of the cycle.

The introduced notation *P* allows us to write the following assertion for the assignment operator $M := M - N$:

$$\{P[M \mapsto M - N]\}\, M := M - N\, \{P\}.$$

Since $M > N \wedge P[M \mapsto M - N] \Rightarrow P[M \mapsto M - N]$ then by the rule of consequence we obtain the following assertion:

$$\{M > N \wedge P[M \mapsto M - N]\}\, M := M - N\, \{P\}.$$

Now we will prove that $M > N \wedge P$ implies $M > N \wedge P[M \mapsto M - N]$. Unfolding $P$, we understand that we need to prove the following implication:

$$M > N \wedge gcd(m,n) = gcd(M,N)[M \mapsto M - N] \Rightarrow$$
$$\Rightarrow gcd(m,n) = gcd(M,N).$$

This can be done in formal arithmetic.

Applying the consequence rule, we obtain the assertion $\{M > N \wedge P\}\ M := M - N\ \{P\}$.

The derivations made can be represented by the following tree (we write the notation of the axiom AS in the conclusion line):

$$C: \frac{AS: \{P[N \mapsto N - M]\}\ N := N - M\ \{P\}}{C: \dfrac{\{\neg(M > N) \wedge P[N \mapsto N - M]\}\ N := N - M\ \{P\}}{\{\neg(M > N) \wedge P\}\ N := N - M\ \{P\}}}$$

Similarly, we obtain the derivation tree for the second assignment operator:

$$C: \frac{AS: \{P[N \mapsto N - M]\}\ N := N - M\ \{P\}}{C: \dfrac{\{\neg(M > N) \wedge P[N \mapsto N - M]\}\ N := N - M\ \{P\}}{\{\neg(M > N) \wedge P\}\ N := N - M\ \{P\}}}$$

The derivations of the two derivations made, namely the assertions

$\{M > N \wedge P\}\ M := M - N\ \{P\}$ and

$\{\neg(M > N) \wedge P\}\ N := N - M\ \{P\}$, are the bases of the IF rule. Applying this rule, we obtain the assertion

$$\{P\}\ \textbf{if}\ M > N\ \textbf{then}\ M := M - N\ \textbf{else}\ N := N - M\ \{P\}.$$

To apply the rule for a cycle, we strengthen the premise to $\neg(M = N) \wedge P$ and, applying the consequence rule, we derive the following assertion:

$$\{\neg(M = N) \wedge P\}$$
$$\textbf{if}\ M > N\ \textbf{then}\ M := M - N\ \textbf{else}\ N := N - M$$
$$\{P\}$$

The derived assertion proves that $P$ is a cycle invariant. In addition, such an assertion is the base of the rule for the cycle operator. Applying this rule, we have:

$\{P\}$

**while** $\neg(M = N)$ **do if** $M > N$ **then** $M := M - N$ **else** $N := N - M$

$\{\neg\neg M = N \wedge P\}$

It is obvious that $\neg\neg M = N \wedge P$ is equal to $M = N \wedge P$.

Now we strengthen the precondition $P$ to

$$M > 0 \wedge N > 0 \wedge M = m \wedge N = n \wedge P.$$

It remains to strengthen the obtained premise to

$$\{M > 0 \wedge N > 0 \wedge M = m \wedge N = n\}..$$

This is obvious, since for $M = m$ and $N = n$ we have that $gcd(m, n) = gcd(M, N)$,

i.e.

$$M > 0 \wedge N > 0 \wedge M = m \wedge N = n \Rightarrow$$
$$\Rightarrow M > 0 \wedge N > 0 \wedge M = m \wedge N = n \wedge P.$$

Note that these conditions are equivalent.

Using the consequence rule, we strengthen the premise to $\{M > 0 \wedge N > 0 \wedge M = m \wedge N = n\}$ and weaken the postcondition to $M = N \wedge M = gcd(m, n)$. We obtain

$\{M > 0 \wedge N > 0 \wedge M = m \wedge N = n\}$

**while** $\neg(M = N)$ **do if** $M > N$ **then** $M := M - N$ **else** $N := N - M$

$\{M = N \wedge M = gcd(m, n)\}$.

Applying the *BE* rule, we obtain the assertion that we had to deduce:

$$\{M > 0 \wedge N > 0 \wedge M = m \wedge N = n\}$$
$$GCD$$
$$\{M = N \wedge gcd(M, N) = gcd(m, n)\}$$

Let us present the derivations in the form of a tree:

$$C:\dfrac{AS:\{P[M\mapsto M-N]\}M:=M-N\{P\}}{C:\dfrac{\{M>N\wedge P[M\mapsto M-N]\}M:=M-N\{P\}}{\{M>N\wedge P\}M:=M-N\{P\}}}$$

$$IF:\dfrac{\begin{array}{|l}C:\dfrac{AS:\{P[N\mapsto N-M]\}N:=N-M\{P\}}{C:\dfrac{\{\neg(M>N)\wedge P[N\mapsto N-M]\}N:=N-M\{P\}}{\{\neg(M>N)\wedge P\}N:=N-M\{P\}}}\end{array}}{C:\dfrac{\{P\}\ \textbf{if}\ M>N\ \textbf{then}\ M:=M-N\ \textbf{else}\ N:=N-M\ \{P\}}{\{\neg(M=N)\wedge P\}}}$$

$$WH:\dfrac{\begin{array}{l}\textbf{if}\ M>N\ \textbf{then}\ M:=M-N\ \textbf{else}\ N:=N-M\\ \{P\}\end{array}}{\begin{array}{l}\{P\}\\ \textbf{while}\ \neg(M=N)\ \textbf{do}\\ \quad\textbf{if}\ M>N\ \textbf{then}\ M:=M-N\ \textbf{else}\ N:=N-M\end{array}}$$

$$C:\dfrac{\{\neg\neg M=N\wedge P\}}{\begin{array}{l}\{P\}\\ \textbf{while}\ \neg(M=N)\ \textbf{do}\\ \quad\textbf{if}\ M>N\ \textbf{then}\ M:=M-N\ \textbf{else}\ N:=N-M\end{array}}$$

$$C:\dfrac{\{M=N\wedge P\}}{\begin{array}{l}\{M>0\wedge N>0\wedge M=m\wedge N=n\wedge P\}\\ \textbf{while}\ \neg(M=N)\ \textbf{do}\\ \quad\textbf{if}\ M>N\ \textbf{then}\ M:=M-N\ \textbf{else}\ N:=N-M\end{array}}$$

$$C:\dfrac{\{M=N\wedge P\}}{\begin{array}{l}\{M>0\wedge N>0\wedge M=m\wedge N=n\}\\ \textbf{while}\ \neg(M=N)\ \textbf{do}\\ \quad\textbf{if}\ M>N\ \textbf{then}\ M:=M-N\ \textbf{else}\ N:=N-M\end{array}}$$

$$BE:\dfrac{\{M=N\wedge M=gcd(m,n)\}}{\begin{array}{l}\{M>0\wedge N>0\wedge M=m\wedge N=n\}\\ GCD\\ \{M=N\wedge M=gcd(m,n)\}\end{array}}$$

To prove total correctness, it is necessary to complicate the rules of logic that would take into account the termination of programs and operators. We will not give details here.

**Example 6.17.** Prove in Floyd-Hoare logic the partial correctness of the program *EXP* calculating $x^n$.

*Solution.* In the following program *EXP* calculating $x^n$, the variables *X* and *N* are used to denote the values of *x* and *n*, respectively, and the variable *R* is used to return the result *r*:

```
EXP ≡
begin
R := 1;
while N > 0 do
   begin
      R := R *X;
      N := N – 1
   end
end
```

First, we formulate an assertion that specifies the correctness of the program *EXP*.

The precondition is the predicate $N \geq 0 \wedge X = x \wedge N = n$, the postcondition is $R = x^n$. Therefore, we need to construct a derivation of the assertion:

$$\{N \geq 0 \wedge X = x \wedge N = n\}\ EXP\ \{R = x^n\}.$$

We prove that the predicate $R * X^N = x^n$ is a loop invariant.

We use the inverse method to construct the preconditions. First, we construct an assertion for the assignment operator $N := N - 1$:

$$\{(R * X^N = x^n)[N \mapsto N - 1]\}\ N := N - 1\ \{R * X^N = x^n\}.$$

We compute $(R * X^N = x^n)[N \mapsto N - 1]$. We obtain the precondition $R * X^{N-1} = x^n$.

Now consider this precondition as a postcondition for the assignment operator $R := R * X$. We obtain the assertion

$$\{(R * X^{N-1} = x^n)[R \mapsto R * X]\}\ R := R * X\ \{R * X^{N-1} = x^n\}.$$

We compute a new precondition $(R * X^{N-1} = x^n)[R \mapsto R * X]$. We obtain the predicate $(R * X) * X^{N-1} = x^n$, which is equal to $R * X^N = x^n$.

Applying the rule for the sequential execution operator, we obtain the assertion

$$\{R * X^N = x^n\} R := R * X; N := N - 1 \{R * X^N = x^n\}.$$

To apply the rule for a cycle, we strengthen the precondition to $N > 0 \wedge R * X^N = x^n$, and by the consequence rule we derive the following assertion:

$$\{N > 0 \wedge R * X^N = x^n\} R := R * X; N := N - 1 \{R * X^N = x^n\}.$$

By the *BE* rule we obtain

$$\{N > 0 \wedge R * X^N = x^n\} \textbf{begin } R := R * X; N := N - 1 \textbf{ end } \{R * X^N = x^n\}.$$

It follows that the predicate $R * X^N = x^n$ is indeed a cycle invariant. Applying the cycle rule, we have

$\{R * X^N = x^n\}$

**while** $N > 0$ **do begin** $R := R * X; N := N - 1$ **end**

$\{\neg(N > 0) \wedge R * X^N = x^n\}$.

Now it remains to derive the formula

$$\neg(N > 0) \wedge R * X^N = x^n \Rightarrow R = x^n$$

This formula is equivalent to the following formula

$$N \le 0 \wedge R * X^N = x^n \Rightarrow R = x^n$$

But this formula is not true. For example, for the values *N* equal to –1, *n* equal to 1, *X* equal to *x*, *R* equal to $X^2$, the formula is false. (Note that the value $X^{-1}$ can be undefined, but we have limited ourselves to cases where all predicates are defined everywhere.)

This situation arises quite often in program verification. The reason is that the chosen invariant $R * X^N = x^n$ is too weak. It must be

strengthened by a condition $N \geq 0$ that will guarantee that the loop terminates with the value *N* equal to 0.

Therefore, consider a new predicate

$$N \geq 0 \wedge R * X^N = x^n$$

and check whether it will be an invariant of the loop.

We construct an assertion for the assignment operator $N := N - 1$:

$$\{(N \geq 0 \wedge R * X^N = x^n)[N \mapsto N - 1]\} N := N - 1 \{N \geq 0 \wedge R * X^N = x^n\}.$$

We calculate $(N \geq 0 \wedge R * X^N = x^n)[N \mapsto N - 1]$. We obtain

$$N - 1 \geq 0 \wedge R * X^{N-1} = x^n$$

Now consider this precondition as a postcondition of the assignment operator $R := R * X$.. We obtain the assertion

$$\{(N - 1 \geq 0 \wedge R * X^{N-1} = x^n)[R \mapsto R * X]\}$$

$$R := R * X$$

$$\{N - 1 \geq 0 \wedge R * X^{N-1} = x^n\}.$$

We calculate a new precondition

$$(N - 1 \geq 0 \wedge R * X^{N-1} = x^n)[R \mapsto R * X]$$

We obtain a predicate $N - 1 \geq 0 \wedge R * X * X^{N-1} = x^n$ that equals $N \geq 1 \wedge R * X^N = x^n$.

Applying the rule for the sequential execution operator, we obtain the assertion

$$\{N \geq 1 \wedge R * X^N = x^n\} R := R * X; N := N - 1 \{N \geq 0 \wedge R * X^N = x^n\}.$$

Now we strengthen the precondition with a predicate $N \geq 0$:

$$\{N \geq 1 \wedge N \geq 0 \wedge R * X^N = x^n\}$$

$$R := R * X; N := N - 1$$

$$\{N \geq 0 \wedge R * X^N = x^n\}.$$

We apply the rule for operator brackets:

$\{N \geq 1 \wedge N \geq 0 \wedge R * X^N = x^n\}$

**begin** $R := R * X; N := N - 1$ **end**

$\{N \geq 0 \wedge R * X^N = x^n\}$.

The obtained assertion $N \geq 0 \wedge R * X^N = x^n$ confirms that it is an invariant of the loop. We apply the corresponding rule, taking into account that it is equivalent $N \geq 1$ to the formula $N > 0$:

$\{N \geq 0 \wedge R * X^N = x^n\}$

**while** $N$>0 **do begin** $R := R * X; N := N - 1$ **end**

$\{\neg N > 0 \wedge N \geq 0 \wedge R * X^N = x^n\}$.

Simplifying the postcondition $N = 0 \wedge R = x^n$, we obtain , which gives the following assertion:

$\{N \geq 0 \wedge R * X^N = x^n\}$

**while** $N$>0 **do begin** $R := R * X; N := N - 1$ **end**

$\{N = 0 \wedge R = x^n\}$.

Consider as the postcondition $\{N \geq 0 \wedge R * X^N = x^n\}$ of the assignment operator $R := 1$:

$\{(N \geq 0 \wedge R * X^N = x^n)[R \mapsto 1]\}\ R := 1\{N \geq 0 \wedge R * X^N = x^n\}$.

Making a substitution, we obtain

$\{N \geq 0 \wedge X^N = x^n\}\ R := 1\{N \geq 0 \wedge R * X^N = x^n\}$.

Recall the premise of the program: $N \geq 0 \wedge X = x \wedge N = n$. This predicate is stronger than the premise $N \geq 0 \wedge X^N = x^n$, that is, the formula $N \geq 0 \wedge X = x \wedge N = n \Rightarrow N \geq 0 \wedge X^N = x^n$ is true. This means that

$\{N \geq 0 \wedge X = x \wedge N = n\}\ R := 1\{N \geq 0 \wedge R * X^N = x^n\}$.

Now we apply the rule for a sequential operator:

$\{N \geq 0 \wedge X = x \wedge N = n\}$

$R := 1$; **while** $N > 0$ **do begin** $R := R * X$; $N := N - 1$ **end**

$\{N = 0 \wedge R = x^n\}$.

Next we apply the BE rule:

$$\{N \geq 0 \wedge X = x \wedge N = n\}\ EXP\ \{N = 0 \wedge R = x^n\}.$$

In the last step of the derivation, it remains to apply the postcondition relaxation rule to obtain the assertion that should have been deduced to prove the partial correction of the program EXP:

$$\{N \geq 0 \wedge X = x \wedge N = n\}\ EXP\ \{R = x^n\}.$$

We will present the derivations in the form of a tree. Since such a tree would be too large to fit on one page, we will present it as two subtrees. First, we will construct a tree (*TW*) of assertion derivation for the loop:

(*TW*):

Now we use the constructed tree (TW) as the second rule base to sequentially execute the assignment and loop statements. We then continue to build the inference tree until we obtain the desired assertion:

$$S:\frac{\begin{array}{l}AS:\{N-1\geq 0\wedge R*X^{N-1}=x^n\}N:=N-1\{N\geq 0\wedge R*X^N=x^n\}\\ \left|AS:\{N\geq 1\wedge R*X^N=x^n\}R:=R*X\{N-1\geq 0\wedge R*X^{N-1}=x^n\}\right.\end{array}}{\{N\geq 1\wedge R*X^N=x^n\}}$$

$$R:=R*X;N:=N-1$$

$$C:\frac{\{N\geq 0\wedge R*X^N=x^n\}}{\{N\geq 1\wedge N\geq 0\wedge R*X^N=x^n\}}$$

$$R:=R*X;N:=N-1$$

$$BE:\frac{\{N\geq 0\wedge R*X^N=x^n\}}{\{N\geq 1\wedge N\geq 0\wedge R*X^N=x^n\}}$$

**begin** $R:=R*X;N:=N-1$ **end**

$$WH:\frac{\{N\geq 0\wedge R*X^N=x^n\}}{\{N\geq 0\wedge R*X^N=x^n\}}$$

**while** $N>0$ **do begin** $R:=R*X;N:=N-1$ **end**

$$C:\frac{\{\neg N>0\wedge N\geq 0\wedge R*X^N=x^n\}}{\{N\geq 0\wedge R*X^N=x^n\}}$$

**while** $N>0$ **do begin** $R:=R*X;N:=N-1$ **end**

$\{N=0\wedge R=x^n\}$.

Now we use the constructed tree (*TW*) as the second rule base to sequentially execute the assignment and loop statements. We then continue to build the inference tree until we obtain the desired assertion:

$$C:\frac{\{(N\geq 0\wedge R*X^N=x^n)[R\mapsto 1]\}R:=1\{N\geq 0\wedge R*X^N=x^n\}}{C:\dfrac{\{N\geq 0\wedge X^N=x^n\}R:=1\{N\geq 0\wedge R*X^N=x^n\}}{\{N\geq 0\wedge X=x\wedge N=n\}R:=1\{N\geq 0\wedge R*X^N=x^n\}}}$$

$$S:\frac{\big|\ (TW)}{\{N\geq 0\wedge X=x\wedge N=n\}}$$

$$R:=1;\textbf{while } N>0 \textbf{ do begin } R:=R*X;N:=N-1 \textbf{ end}$$

$$BE:\frac{\{N=0\wedge R=x^n\}}{\{N\geq 0\wedge X=x\wedge N=n\}\ EXP\ \{N=0\wedge R=x^n\}}$$

Note that the rules apply directly to programs, and not to their semantic terms. This allows the use of classical first-order logic to record pre- and post-conditions. In the case of more complex programs, it is advisable to apply the rules to semantic terms and use logics oriented to these terms – compositional-nominative logics.

Additional issues of semantics are considered in [NikSh2013].

### *6.7. Questions and exercises*

The following tasks are to be performed using sequent calculus for propositional logic and predicate logics.

1. Prove the consequence statement:
   1) $a,\ a\to b\models b$;
   2) $a\to b\ ,\ \neg b\models\neg a$;
   3) $a\vee b,\ \neg a\models b$;
   4) $a\vee b,\ a\to b\models b$;
   5) $a\to b,\ a\wedge\neg b\models b$;
   6) $a\vee b,\ a\to c,\ b\to d\models c\vee d$;
   7) $\neg a\vee b,\ \neg b\vee\neg c\models a\to\neg c$;
   8) $\neg(a\vee b)\models\neg a\vee c$;
   9) $a,\ c\to\neg(a\vee b)\models\neg c$;

10) $(a \to b) \to c,\ a \to (b \to c) \models b \to c$;
11) $a \vee \neg c,\ a \to d,\ b \to c,\ \neg b \to d \models d$;
12) $a \to (b \vee c),\ d \to a \models (\neg b \wedge \neg c) \to \neg d$.

2. Determine whether the following consequence statement is correct:

1) $(a \to b) \to c \models a \wedge b \vee c$;
2) $a \vee b \vee c \models a \to b \to c$;
3) $a \to b,\ a \vee c \models a \wedge c \to a \wedge b$;
4) $a \to b,\ a \to c,\ a \models b \vee c$;
5) $a \to b,\ c \wedge a \models c \wedge b$;
6) $a \to b,\ c \wedge a \models c$;
7) $a \wedge b,\ \neg a \vee b \models \neg b$;
8) $a \vee b,\ a \to c,\ b \to c \models \neg c \to \neg a$.

3. Find logical consequences from the premises:

1) $a,\ a \to b$;
2) $a \to b\ ,\ \neg b$;
3) $a \vee b,\ \neg a$;
4) $a \vee b,\ a,\ \neg b$;
5) $a \to b \to c\ , b \to c$;
6) $a \vee b,\ a \to c,\ b \to d$.

4. Find the premises whose logical consequences are the following formulas:

1) $a \to b$;
2) $a \wedge b$;
3) $a \to b \wedge c$;
4) $a \vee b \to a \wedge b$.

5. Are the following formulas always true:

(1) $(A(x) \to \forall x B(x)) \to (\forall x A(x) \to B(x))$;

(2) $(A(x) \to \exists x B(x)) \to \exists x (A(x) \to B(x))$ ;

(3) $(\exists x A(x) \to B(x)) \to (A(x) \to \exists x B(x))$ ;

(5) $(\forall x A(x) \to B(x)) \to (A(x) \to \forall x B(x))$ ;

(6) $\exists x (A(x) \to B(x)) \to (A(x) \to \forall x B(x))$ ;

(7) $(A(x) \to \forall x B(x)) \to \exists x (A(x) \to B(x))$ ;

(8) $\exists x A(x) \wedge B(x) \to \exists x (A(x) \wedge B(x))$ ;

(9) $\exists x A(x) \wedge B(x) \to \forall x (A(x) \wedge B(x))$ ;

(10) $\exists x (A(x) \wedge B(x)) \to \exists x A(x) \wedge B(x)$ ;

(11) $\exists x A(x) \vee B(x) \to \exists x (A(x) \vee B(x))$ ;

(12) $\exists x (A(x) \wedge B(x)) \to \exists x A(x) \vee B(x)$ ;

(16) $A(x) \wedge \forall x B(x) \to \forall x (A(x) \vee B(x))$ .

6. Prove the equivalences:

(1) $\neg \forall x\, A(x) \leftrightarrow \exists\, x \neg A(x)$;

(2) $\forall x\, \forall y\, A(x, y) \leftrightarrow \forall y\, \forall x\, A(x, y)$;

(3) $\exists x\, \exists y\, A(x, y) \leftrightarrow \exists y\, \exists x\, A(x, y)$;

(4) $\exists x\, (A(x) \vee B(x)) \leftrightarrow \exists x\, A(x) \vee \exists x\, B(x)$;

(5) $\forall x\, (A(x) \wedge B(x)) \leftrightarrow \forall x\, A(x) \wedge \forall x\, B(x)$.

## Chapter 7. Intensionality in compositional logics

Intuitively, intensional aspects represent properties of the notion under investigation and extensional aspects represent individuals that have intensional properties. The intensional can be treated as finitization of the category Universal on the scientific level and the extensional is finitization of the category Individual. We also propose to call finitization of the category Particular as 'especial' because this word is more formal than 'special' or 'particular'.

In this chapter intensional and extensional aspects in logic will be considered.

### *7.1. Intensional and extensional aspects in logic*

Mathematical basis of logic is formed by set theory, universal algebra, and computability theory. Many important and useful results for logic were obtained on this basis. Still, some discrepancies between above mentioned disciplines and problems of logic can be also admitted. They concern questions of

- defining data structures on various levels of abstraction,
- computability of functions over such data,
- processing of data with incomplete or fuzzy information,
- construction of logics oriented on information processing, etc.

1Such discrepancies require additional efforts in modeling problems with existing mathematical formalisms. Therefore, it is reasonable to formulate a problem of developing own, more adequate mathematical foundations for logic. But what should be the starting point of such development?

Analysis of existing mathematical formalisms shows that they are constructed on a set-theoretic platform. But its main notion – the notion of set – is explicated in extensional style. This style is supported by the very first axiom of set theory – the extensionality axiom: two sets are equal if they consist of the same elements [Bourbaki1968].

N. Bourbaki in his numerous treatises aimed to write a thorough unified account of all mathematics based on extensional set theory. At that period the extensional approach played a positive role permitting to specify formally many properties of mathematical objects. But we can see now more and more facts when a pure extensional orientation becomes restrictive for further development of informatics.

Thus, we propose to add explicit intensional component to notions and construct them in integrity of intensional and extensional aspects. Here the intension of a notion (of a concept) means properties which specify that notion, and the extension means objects which fall under the notion, i.e. have the properties specified by the notion intension. Intension and extension of a notion we consider as projections of categories *Universal* and *Individual* respectively.

We should admit that this proposal is not new. The distinction between intensional and extensional aspects of a notion was known from ancient times. Aristotle in his *Posterior Analytics* already specified this distinction though he did not use explicitly the above terms. Many logicians since that time examined the questions of intension/extension dichotomy. A second wind to these investigations was given by G. Frege with his famous *meaning triangle* and R. Carnap with his *intensional/extensional investigations*. Though the dichotomy under discussion was studied not only in logic, but also in semiotics, linguistics, and informatics. In its branches related to artificial intelligence, data and knowledge bases, semantic web, etc., the intensional aspects now play an important role. But in formalized (mathematical) theories intensional aspects are still used very restrictively.

The above presented considerations advocate the following principle:

a notion should be presented by the triad *notion intension – notion extension – integrity of intension and extension; the intension in this integrity play a leading role* (*the principle of triadic model of a notion*).

This principle may be considered as an enhancement of Frege's meaning triangle.

At the formal level of foundations, the notions, constructed at the previous level, are specialized in order to get their reasonable formalization. This formalization should take into account intensional and extensional notion aspects. This level is important because formal notions provide a basis for automatization of various phases of information processing.

Having specified a three-level structure of foundations and main principles of theory development, we can now consider basic notions of logic at the conceptual level. As the name shows, one of the main notions is the notion of *information*. It can be formalized in various aspects, but the most important ones are aspects represented by the philosophical categories of *Form* and *Content*. This understanding is supported by the etymology of 'information', derived from Latin '*informare*': "to give form to". Forms of information which are relatively independent of its content we call *data*.

Thus, we get the initial part of the well-known

"data – information – knowledge – wisdom"

hierarchy (DIKW-hierarchy) [Ack1989].

These concepts are the building blocks of logic; and explications should be developed for all these notions. This is a difficult challenge, therefore it is reasonable to start with the notion of data which lies in the base of DIKW-like hierarchies, is relatively simple, and is an appropriate subject for further formalization. The word 'data' is used here as a plural and as a singular noun.

Among many characteristics of data (see [Zins2007] for a detailed discussion) we choose "data as manipulable objects"

[Hey2004]. This treatment is a projection on conceptual level of the category of *object* taken in the praxeological aspect.

So, our abstract understanding of the notion of data includes three moments only:

1) data are regarded as a form of information content;

2) data are (relatively) independent from information content;

3) data are manipulable objects.

These moments demonstrate that we treat data in a very broad sense. The principle of development from the abstract to the concrete suggests that other data characteristics should be introduced on the later stages of development.

Previously (Chapter 4), we defined data in the extensional manner. In this chapter, according to the dialectical triad scheme, we will treat data as integrity of their intension and extension, thus we enrich traditional understanding of data with intensional component [Nik2001].

Obtained data are called *intensionalized data*. Let us admit that there is an analogy with the notion of *typed data*, but the latter is usually understood in the extensional sense while we aim to emphasize intensional features of the approach.

### *7.2. Intensionalized data*

To simplify formalization of the notion of intensionalized data, we start with the most abstract understanding of data as some objects (keeping in memory that on later stages data should be considered as manipulable objects representing information content). The first step in objects explication is classification of intensions which can be prescribed to objects. Such classification can be made with respect to various criteria. Our three-level activity-based approach suggests to develop a classification induced by some categories. As such we choose categories *Whole* and *Part* which are among the first categories revealing the category of *Essence* [Hegel1969]. Thus, we intro-

duce two different intensions $I_W$ and $I_P$: object as a whole (unstructured object) and object with parts (structured object) respectively.

Further development of $I_W$ is done in accordance with categories of *Abstract* and *Concrete*. One of projections (restrictions) of these categories on the conceptual level describes abstract as less informative than concrete. In extreme cases, an object can be regarded as a "black box" (intuitively it means that nothing is "visible", and therefore nothing is known about object) or as a "white box" (everything is "visible" and recognizable). Thus, we articulated new intensions $I_{WB}$ and $I_{WW}$ respectively. An intermediate intension is denoted by $I_{WBW}$ ("black or white box"). The introduced intensions describe the main possibilities to treat object as a whole.

To come to richer intensions, we should treat objects as structured (with intension $I_P$). In this case we get a triad: *whole – part – structure*, where structure is the synthesis of categories *Whole* and *Part*. Now we will invent properties (intensions) of object structures. The development principle stimulates us to start with simple structures. Simplicity means that all parts of an object are recognized and fixed. In this case each part can be regarded as a whole. Relations within the object are also recognized and fixed. The next question is: what intensions can be prescribed to the parts and relations? Being the wholes, parts can have intensions of black and/or white boxes. Should it be allowed for relations to have the same three intensions? It is not reasonable to do this at the first stages of data development; therefore, we prescribe to relations intensions of white boxes only. The above specification of object structure permits to call it *hard structure*. Thus, we divide the intension $I_P$ into two sub-intensions $I_{PH}$ and $I_{PS}$ specifying objects with hard and soft structures respectively. Here we restrict ourselves by studying objects with prescribed intension $I_{PH}$ only which is simpler than $I_{PS}$.

We continue with $I_{PH}$ classification that is caused by possible relations between object parts. Such relations are classified along the line *tight–loose*. Loose relations mean that parts are not connected with each other (in Hegel's words, are indifferent to each other);

tight relations mean that parts are connected. Thus, new intensions $I_{PHL}$ and $I_{PHT}$ are articulated.

Parts of objects with intension $I_{PHL}$ and $I_{PHT}$ are usually called *elements* and *components* respectively. Considering elements as wholes, we can treat them with intensions of black and/or white boxes. Three new intensions which are sub-intensions of $I_{PHL}$ stem from this: $I_{PHLB}$, $I_{PHLW}$, and $I_{PHLBW}$ .

It is worth to discuss objects (data) with these intensions in more detail. Objects with intension $I_{PHLB}$ should be regarded as collections of black boxes because they consist of clearly separated elements with unknown interior (content) that have no relations with each other. Such objects we call *presets*. For example, buying several tickets of an instant lottery someone gets a preset because surfaces of the tickets are covered by opaque material making them black boxes. Collections of elements which are white boxes (intension $I_{PHLW}$) are called *explicit sets*. Collections with intension $I_{PHLBW}$ contain "black" and "white" elements (*mixed presets*). For example, when someone comes at a party he first classifies people as known to him (white elements) or unknown (black elements). Here we should note that richer intensions than $I_{PHLB}$ and $I_{PHLBW}$ can allow extracting additional information from "black" elements (and in this case they become "white" or "gray" elements).

Thus, we have articulated three kinds of objects: presets, sets, and mixed presets. Now just sets serve as a basis for formalization of informatics notions, but it seems reasonable to use also the notions of preset and mixed preset for this purpose. Their importance can be substantiated by the necessity of defining data at various abstraction levels that cannot be adequately captured by the extensional notion of set. Therefore, in is not strange that numerous attempts were made for constructing set theory without extensionality (see, for example, [App2009]). We stop further classifications of objects with intension $I_{PHL}$ ($I_{PHL}$-objects) and start classifying $I_{PHT}$-objects.

Components of $I_{PHT}$-objects are in some way related to each other, so, contrary to the elements of $I_{PHL}$-objects, the components are not allowed to permute freely with each other within $I_{PHT}$-objects. Ex-

amples are: lists, trees, graphs, arrays, etc. It seems (and it is true) that a lot of useful kinds of $I_{PHT}$-objects can be considered. This fact poses a question what sub-intension of $I_{PHT}$ should be introduced first? And again, we appeal to the principle of development which advises to start with the simplest relation between components. Such a relation connects only two components. In this case these components look as opponents, and the relation should link them, thus making their synthesis. From this follows that it is reasonable to treat the first component as a black box, the second as a white box, and their synthesis as a "dipole" consisting of the black and white boxes. To make these abstract considerations more concrete we should involve practical observations of activities in the information domain which prompts us that the white box is a *name* of the black box; and their relation is a *naming* (*nominative*) relation. Thus, described objects are presets whose elements are named values. Such objects may be called *nominats* with corresponding intension as $I_{ND}$. In Slavic languages the term 'nominat' has two different meanings: a naming expression or a value of such expression. Our proposal unites these meanings, because nominat is a unity of names and values. Still, we will use here the term *nominative data*.

Traditionally, notations of functional style are chosen to represent nominative data. For example, a nominative data with names $v_1$, …, $v_n$ and values $a_1$, …, $a_n$ respectively, is denoted by

$$[v_1 \mapsto a_1, \ldots, v_n \mapsto a_n].$$

If the values themselves are nominative data, then we get the notion of hierarchic nominative data; for example

$$[v_1 \mapsto [u_1 \mapsto b_1, \ldots, u_k \mapsto b_k], \ldots, v_n \mapsto [t_1 \mapsto c_1, \ldots, t_m \mapsto c_m]]$$

is a 2-level nominative data.

It is important to admit that nominative data can model the majority of data structures used in informatics (see Chapter 4). For example, a set $\{e_1, \ldots, e_m\}$ can be represented as $[1 \mapsto e_1, \ldots, 1 \mapsto e_m]$, where 1 is a standard name which have different values $e_1$, …, $e_m$; a tuple $(e_1, \ldots, e_m)$ can be represented as $[1 \mapsto e_1, \ldots, m \mapsto e_m]$ with 1,…, m as standard names; a sequence $<e_1, \ldots, e_m>$ can be represented as

$$[1 \mapsto e_1, 2 \mapsto [\ \ldots, 2 \mapsto [1 \mapsto e_m, 2 \mapsto \varnothing_n] \ldots]],$$

where 1, 2 are standard names and $\varnothing_n$ is the empty nominative data.

Summing up, we can conclude that the developed notion of intensionalized data can represent data structures used in informatics, and besides, this representation looks richer and more adequate than traditional set-theoretic representation.

Here we have concentrated only on classification of object intensions ignoring operations with objects. To come back to the initial treatment of data as manipulable objects, we should describe operations allowed for data with different intensions. Actually, it means that we should try to construct basic computability theory for intensionalized data.

### *7.3. Computability over intensionalized data*

We begin with general considerations about traditional computability. Such computability is usually understood as computability of *n*-ary functions defined on integers or strings. It may be called Turing computability. In the light of our investigations traditional computability does not pay much attention to variety of data intensions, because it concentrates on computability over integers (or strings) which have fixed intensions. But in informatics other data structures with different intensions are also used; therefore, for these structures a new notion of computability is required [Nik2001n].

The computability problem is not the only aim of our investigations. Now it is generally recognized that information systems should be developed successively from abstract specifications via more concrete representations up to detailed implementations in chosen programming languages. And it is important to connect computability with stages of system development. We intend to introduce such a unified notion of computability that can be applied to every stage of system de-

velopment and can be easily transformed when moving from stage to stage. Such a kind of computability should be applicable to data structures of different abstraction levels and is called *abstract computability*. A partial case of this computability, oriented on intensionalized data, is called *intensionalized computability*. The idea behind it is the following: for data processing it is allowed to use only those operations that conform to their intensions. Thus, intensionalized computability is *intensionally restricted computability*. In fact, such computability is a *relative computability* – relative to data intensions. Usually it is required that data have *finite structures*; a corresponding data intension we denote by $I_{PHF}$.

There is a difficulty in defining intensionalized computability that is caused by the fact that for finite structured data with intension $I_{PHF}$ we do not have precise definitions of their components and relations between components; thus, precise definition of computability is not possible. Of cause, we can introduce data with precisely described sub-intensions of $I_{PHF}$ (like intensions for tuples, list, trees, etc.) and then define computability for such specific intensionalized data. But in this case we will not get a unified computability theory for intensionalized data. To overcome this difficulty, we propose to apply the method of reduction of intensionalized computability to traditional Turing computability. To do this we will first define a special form of finite structured data with a fixed intension, and then reduce data with other intensions to this special form.

Let $D$ be a class of data with intension $I_D$. Such class is also denoted as $[I_D, D]$, data from this class are called $I_D$-intensionalized data or simply $I_D$-data. Assume that we treat data from $D$ as finite structured data. Our intuitive understanding of a such data is the following: any such data $d$ consists of several basic (atomic) components $b_1, ..., b_m$, organised (connected) in a certain way. If there are enumerably many different forms of organisation, each of these data can be represent-

ed in the (possibly non-unique) form ($k$,<$b_1$, ..., $b_m$>), where $k$ is the *data code* and the sequence <$b_1$, ..., $b_m$> is the *data base*. Data of this form are called *natural data* [Nik2001n]. More precisely, if $B$ is any class and *Nat* is the set of natural numbers, then the class of *natural data* over $B$ is the class $Nat(B) = Nat \times B^*$. We use the term 'class' for collections of intensionalized data; term 'set' is used for collections which intensions are sub-intensions of sets. As finite structured data can have different representations, we should introduce multi-valued functions for constructing such representations. Function $f$ is *multi-valued* (non-deterministic) if being applied to the same input data $d$ it can yield different results during different applications to $d$ (i.e., function's graph is not a functional relation). To avoid complex notations with subscripts, to will denoted a class of partial multi-valued functions over $D$ as $D \to\to D$. A multi-valued function is *injective*, if it yields different values on different arguments.

Now we are ready to give the formal definition of a class of intensionalized data with some intension $I_D$ which is a sub-intension of $I_{PHF}$. A class $D$ is called a class *of finite structured data*, if a class $B$ and a total multi-valued injective mapping *nat*: $D \to\to Nat(B)$ are given. This mapping *nat* is called the *naturalization* mapping. Naturalization mapping is actually an *analysing* mapping: it finds in a data $d$ its components and their interrelations according to the properties of data prescribed by its intension. Dually to *nat* we introduce *denaturalization* mapping *denat* which reconstructs (*synthesizes*) data of class $D$ from natural data. For simplicity's sake we assume that $denat = nat^{-1}$. Denaturalization mapping is a partial single-valued mapping. Naturalization and denaturalization mapping are also called *concretization* and *abstraction* mappings respectively.

Introduction of naturalization mapping is a crucial moment for defining intensionalized computability. This mapping can

be regarded as a formalization of data intension; and this enables us to reduce an intuitive notion of intensionalized computability over $D$ with intension $I_D$ to formally defined *natural computability* over $D$. The latter is then reduced to a new special computability over $Nat(B)$ that is called *code computability*. To define this type of computability we should recall that in a natural data the code collects all known information about data components. Thus, code computability should be independent of any specific manipulation (processing) operations of the elements of $B$ and can use only information that is explicitly exposed in the natural data. The only explicit information is the data code and the length of the data base. Therefore, in code computability the data code plays a major role, while the elements of the data base are treated as black boxes which virtually do not affect the computations. These elements may be only used to form the base of the resulting data. To describe the code of the resulting data and the order in which elements of the initial base are put into the base of resulting data, a special function of type $Nat(B)\to\to Nat(B)$ should be defined. Such a function is called *index-computable*. These considerations lead to the following definition.

A function $g$: $Nat(B)\to\to Nat(B)$ is called *code-computable* if there exists an index-computable multi-valued function $h$: $Nat^2\to\to Nat \times Nat^*$ such that for any $k$, $m\in Nat$, $b_1,...,b_m\in B$, $m\geq 0$, we have $g(k,<b_1,...,b_m>) = (k',<b_{i_1}, ..., b_{i_l}>)$ if and only if $h(k,m) = (k',<i_1,...,i_l>)$, $1\leq i_1\leq m,..., 1\leq i_l \leq m$, $l\geq 0$. If one of the indexes $i_1,...,i_l$ lies outside the interval $[1, m]$, or $h(k,m)$ is undefined, then $g(k,<b_1,...,b_m>)$ is also undefined.

In other words, in order to compute $g$ on $(k,<b_1,...,b_m>)$, we have to compute $h$ on $(k,m)$, generate a certain value $(k',<i_1,...,i_l>)$, and then try to form the value of the function $g$ by selecting the components of the sequence $<b_1,...,b_m>$ pointed to by the indexes $i_1,...,i_l$.

It is clear, that index computability of $h$: $Nat^2 \to\to Nat \times Nat^*$ may be reduced by traditional methods of recursion theory to computability of a certain partial recursive function $r$:$Nat \to\to Nat$.

We are ready now to give the main definition of this section. A function $f$: $D \to\to D$ is called *naturally computable* (with respect to given $B$ and *nat*) if there is a code-computable function $g$: $Nat(B) \to\to Nat(B)$ such that $f = denat \circ g \circ nat$ .

The class of all naturally computable functions is denoted by $NatComp(D, B, nat)$.

Thus, intensionalized computability is defined via a sequence of the following reductions: intensionalized computability – natural computability – code computability – index computability – partial recursive computability. Analysing the definitions, we can also conclude, that natural computability as a generalization (relativization) of enumeration computability. In fact, for $B = \varnothing$ code computability is reduced to partial recursive computability on *Nat*, and natural computability is reduced to enumeration computability (with respect to *nat*). Therefore, the notions of code and natural computability defined above are quite rich.

Having defined the notion of natural computability, we can now construct algebraic representations of complete classes of naturally computable partial multi-valued functions for various kinds of intensionalized data. Here we give without details only few examples. We start with the simplest case.

Let $D$ be a preset with prescribed intension $I_{PS}$ (=$I_{PHLB}$). It means that nothing is known about its elements. This treatment can be formalized by the naturalization mapping nat[$I_{PS}$]: $D \to\to Nat(D)$ such that $nat[I_{PS}](d)=(0,<d>)$ for every $d \in D$. To define the complete class of naturally computable functions over $[I_{PS}, D]$, we have to describe all index-computable function of the type $h$: $Nat^2 \to\to Nat \times Nat^*$. It is easy to understand that under the naturalization mapping

*nat*[$I_{PS}$] we need to know the results of index-computable function only on the element (0,1). On this input data an index-computable function 1) can be undefined, 2) can yield (0, 1), or 3) can make non-deterministic choice between being undefined and yielding (0, 1).

These three cases induce the following functions of type $D \to\to D$: 1) the everywhere undefined function *und*, 2) the identity function *id*, and 3) the non-deterministic function *und-id* such that *und-id*(*d*) is undefined or is equal to *d*.

It means that the following result was proved: *the complete class of naturally computable partial multi-valued functions over* $I_{PS}$*-intensionalized preset D consists of functions und, id, and und-id.*

In other words, the three functions defined above are the only computable function over "black box" intensionalized data.

The next example describes computability over subclass of hierarchic nominative data. This subclass *NAD*(*V,W*) is called the class of *named data* and is defined inductively on the basis of a finite set of names $V = \{v_1, \ldots, v_m\}$ and a preset of basic values *W*:

1) If $w \in W$, then $w \in NAD(V,W)$,
2) If $v_1,\ldots,v_n$ are pairwise distinct names from *V*, $d_1, \ldots, d_n$ are from *NAD*(*V,W*), then $[v_1 \mapsto d_1, \ldots, v_n \mapsto d_n]$ belongs to *NAD*(*V,W*).

The intension of such data is denoted by $I_{NAD}$. This understanding of *NAD*(*V,W*) can be represented by the naturalization mapping *nat*[$I_{NAD}$]: $NAD(V,W) \to\to Nat(W)$ which is defined inductively as follows:

1)if $d \in W$, then nat[$I_{NAD}$](d)=(c(0,0),<d>);

2)if $d = [v_{i_1} \mapsto d_1,\ldots, v_{i_n} \mapsto d_n]$, $i_1<\ldots<i_n$, $n \geq 0$,

$nat[I_{NAD}](d_j)=(k_j,<b_{j1}, \ldots, b_{jl_j}>)$, $1 \leq j \leq n$, then

$$nat[I_{NAD}](d) = (c(1, c(n,c(k'_1,...c(k'_n,0)...))),<b_{11},...,b_{1l_1}, ...,b_{n1}, ..., b_{nl_n}>),$$

where $c$: $Nat \times Nat \to Nat$ is the Cantor's pairing function; $k'_j = c(i_j,c(k_j, l_j))$, $1 \le j \le n$.

Having defined the naturalization mapping for $[I_{NAD}, NAD(V,W)]$, we obtain the class $NatComp(D, B, nat[I_{NAD}])$ of all naturally computable functions over the class $NAD(V,W)$. As the basic functions from this class we choose operations [15] of *naming* $\Rightarrow v$, *denaming* $v\Rightarrow$, and *checking* $v!$ with name $v \in V$ as a parameter; we also use non-deterministic *choice* $\diamondsuit$ which on $d$ yields $d$ or $\varnothing_n$. The main operations over this class of functions (we call them *compositions*) are *multiplication* $\circ$ (functional composition), *conditional iteration* $*$ (while–do), and *overriding* $\nabla$.

The following theorem which gives an algebraic description of the class $NatComp(D, B, nat[I_{NAD}])$ can be proved:

*the complete class of naturally computable partial multivalued functions over the* $I_{NAD}$*-intensionalized class of named data NAD(V,W) coincides with the class of functions obtained by closure of functions* $\Rightarrow v$, $v\Rightarrow$, $v!$, *and* $\diamondsuit$ *under compositions* $\circ, *,$ *and* $\nabla$ $(v \in V)$.

The last example will describe computability over the $I_{SEQ}$-intensionalized class $Seq(B)$ of sequences constructed hierarchically over a preset $B$. The structure $Seq(B)$ has been investigated in different works. We shall use the notations of [Bac1978]. The following functions are introduced: *first*, *tail*, *apndl*, and *is-atom*. Also, we need a composition, called *construction*: $[f,g](d) = <f(d), g(d)>$ ($d$ belongs to $Seq(B)$, $f$ and $g$ are functions over $Seq(B)$).

The following theorem can be proved:

*the complete class of naturally computable partial multivalued functions over the* $I_{SEQ}$*-intensionalized class Seq(B) coincides with the class of functions obtained by closure of*

*functions first, tail, apndl, is-atom, and ♢ under compositions ∘, *, and [].*

Having introduced the notion of intensionalized computability, we actually defined those operations which are allowed to apply to such data. Thus, combining definitions of intensionalized data with definitions of computable functions over such data, we made explication of data as manipulable objects. To reason about intensionalized data, we should develop special logics oriented on such data.

### *7.4. Predicate logics over intensionalized data*

The main idea of developing logics over intensionalized data consists in defining such logical constructs (connectives, quantifiers, etc.) that conform to the data intensions. It means that these logical constructs (to be semantically explicated as compositions of predicates) should use only such data information that is specified by data intensions.

To make these intuitive considerations more strict, we start with constructing a semantic base for intensionalized logics.

Let $[I_D, D]$ be a class of intensionalized data. A class of partial functions $P = D \rightarrow Bool$ is called a class of *partial predicates* over $D$ ($Bool = \{T, F\}$). Operations over $P$ are called *predicate compositions*. Let us admit that we do not restrict predicates by data intensions. This is necessary in order to have a wider class of models for logics. But data intensions should restrict the class of predicate compositions; from this stem the main problem of logic development: how to define compositions which are intensionally restricted by $I_D$?

We will define such compositions according to the principle of development from the abstract to the concrete. Therefore, we should start with the most abstract intension $I_B$ (“black box” data). The basic compositions defined in this case are

compositions of predicate disjunction $\vee$ and negation $\neg$. We define these compositions in the style of strong Kleene's connectives.

Let $p$ and $q$ be predicates, $d$ be from $D$. Then $(p \vee q)(d)$ is defined and equal to $T$ if $p(d)$ or $q(d)$ is defined and equal to $T$; is defined and equal to $F$, if both $p(d)$ or $q(d)$ are defined and equal to $F$; and is undefined in all other cases. The value $\neg p(d)$ is a dual to the value of $p(d)$. Other propositional compositions can be defined analogously.

From these definitions we see that logics over $[I_B, D]$ are just different variants of propositional logics (partiality should be also taken into account). A semantics base of such logics are predicate algebras of the form $<D \rightarrow Bool, \vee, \neg >$. Properties of such algebras will specify calculi for our logics.

We will not describe logics over classes of data with intensions $I_W$ and $I_{BW}$, because the intension $I_W$ specifies only one concrete class of data, and its logic is only a logic of this class (singular logic); logics over $I_{BW}$-intensionalized data are combinations of propositional and singular logics.

The next data intension we consider here is the intension of (infinite) flat nominative data $I_{ND}$ which is a sub-intension of $I_B$. Data, intensionalized with $I_{ND}$, have a form $[v_1 \mapsto a_1, v_2 \mapsto a_2, \ldots]$. In traditional logic names $v_1, v_2, \ldots$ are called individual variables, and data from $D$ are called variable assignments, variable valuations, etc. In this case $D={}^VA$, where $V$ is a set of individual variables, $A$ is a class of individual values, and ${}^VA$ can be considered as a class of partial function from $V$ to $A$. Predicates from ${}^VA \rightarrow Bool$ are called *quasi-ary predicates.* At the level of $I_{ND}$-data, additionally to propositional compositions, we can define a new composition of *renomination* $R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n}$. Given a predicate $p$ and data $d$ the value of $R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n}(p)(d)$ is equal to the value of $p(d')$, where $d'$ is obtained from $d$ by assigning to variables $v_1, \ldots, v_n$ values of variables

$x_1$, …, $x_n$ respectively. Note, that renomination composition (primarily in syntactical aspects) is widely used in classical logic, lambda-calculus, and specification languages like Z-notation, B, TLA, etc. The obtained logics are called renomination logics (quantifier-free logics). Their semantics base are predicate algebras of the form $<{}^{V}A\to Bool,\ \vee,\ \neg,\ R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n}>$. Properties of such algebras will specify calculi for renomination logics.

To introduce first-order logics we should specify a new data intension $I_{NDQ}$. This intension allows an exhaustive search within the class *A*. It permits to introduce quantification compositions $\exists x$ and $\forall x$ ($x$ is a variable). The value of $\exists x p(d)$ is defined and equal to *T*, if there is *d'*, differing from *d* only in the value of *x*, such that $p(d')$ is defined and equal to *T*; is defined and equal to *F*, if for all such *d'* the value $p(d')$ is defined and equal to *F*; and is undefined in all other cases. The composition of universal quantification is defined in a similar way. The semantics base of first-order logics are predicate algebras of the form

$$<{}^{V}A\to Bool,\ \vee,\ \neg,\ R^{v_1,\ldots,v_n}_{x_1,\ldots,x_n},\ \exists x>.$$

To preserve properties of classical first-order logic we should restrict the class ${}^{V}A\to Bool$ of quasi-ary predicates. This restriction stems from the fact that predicates of first-order logic, being defined on some data, are also defined with the same value on all extensions of this data. Such quasi-ary predicates are called *equitone*. We also introduce different variations of equitone predicates such as maxitotal (necessarily defined on maximal data), local-equitone (equitone for finite extensions only), and equicompatible (extensible to equitone predicates). Logics based on equitone and maxitotal equitone predicates are the "closest" generalizations of classical first-order logic that preserve its main properties. These logics are called *neoclassical logics*. For all these logics corresponding calculi were con-

structed; their *soundness and completeness* were proved. All necessary mathematical details can be found in [NikSh2013].

The last class of logics considered here are logics over hierarchic nominative data with intension $I_{NDH}$. Corresponding logics will use composite names of the form $x_1.x_2. \ldots .x_n$ as parameters of renomination and quantification compositions. For this case their definitions should be redefined to take into account hierarchic structure of data.

Similar analysis of intensional aspects of program compositions can be made.

Summing up, we can conclude that the notion of intensionalized data

1) permits to construct new kinds of semantically-based predicate logics oriented on such data (intensionalized logics), and

2) gives possibility to explain the origination of classical logics as logics oriented on "black box" data (propositional logics) or on flat nominative data (first-order logics).

### *7.5. Questions and exercises*

1. What are intensional aspects of a notion?
2. What are extensional aspects of a notion?
3. Finitizations of what philosophical category is the notion of intension?
4. Finitizations of what philosophical category is the notion of extension?
5. Write an essay about Frege's triangle and its applications in logic.
6. Why is mathematics based on the extensional approach?
7. Formulate the axiom of extensionality of set theory.
8. How are intensional definitions are used in mathematics and logic?
9. Write an essay on non-classical set theories.
10. Formulate the principle of intensional-extensional integrity.
11. What is DIKW-hierarchy?
12. What is the intuitive understanding of data?

13. What is the fundamental aspect of data?
14. How is the concept of data related to the concept of information?
15. What are intensionalized data?
16. What categories are definitions of intensional data associated with?
17. How are intensions defined in intensionalized data?
18. How do intensions develop in intensionalized data?
19. What is a preset?
20. Describe the concept of black, white, and gray boxes.
21. How is the concept of traditional computability introduced?
22. Write an essay on non-classical models of computability.
23. What are finite data structures?
24. What are natural data?
25. How are intensions formalized for the intensionalized data?
26. What are naturalization and denaturalization functions?
27. How is index-computability defined?
28. How is code-computability defined?
29. How is natural computability defined?
30. How can the class of naturally computable functions over presets be specified?
31. How can the class of naturally computable functions over intensionalized named data be specified?
32. How can the class of naturally computable functions over intensionalized sequences be specified?
33. How to define intensions for predicate compositions?
34. What are the levels of predicate compositions with respect to their intensions?
35. How do intensions manifest themselves in composition logics?

## Conclusions

Compositional logics form a broad area of mathematical logic. They are in the theoretical basis of artificial intelligence, computer science, and software engineering. They are widely used for specification and verification of software systems.

The main topics considered in the textbooks are as follows:

- Methodological basis of compositional logics was developed according to the principles and schemes of three-level (philosophical, scientific, and mathematical) activity-based approach;
- The hierarchy of compositional logics was successively development starting from abstract logic via propositional and predicate logics up to modal logic. This hierarchy reflects levels of activity;
- Semantic base of compositional logics was specified by predicate algebras (especially by algebras of quasiary predicates), adequately formalizing semantics of predicate languages. Various types of such algebras and problems of formulas satisfiability are considered.
- Nominative data treated as an universal data structure capable to model data structure used for subject domain modelling were studied;
- Algebras and logics used to specify properties of software systems were presented;
- Various deductive systems, especially sequent calculi, for the logics described earlier, were presented; numerous examples were considered;
- Intensionality in compositional logics was discussed; intensionalized computability was presented. Contrary to extensional approach to formalization of logics, intensional approach emphasizes importance of general (universal) aspects in logic.

**References**


[Ben-Ari2022] Mordechai Ben-Ari. Mathematical Logic for Computer Science, Springer-Verlag London, 2012.

[Hegel1969] G. W. F. Hegel, Hegel's Science of Logic / translated by Miller A. V., London: G. Allen & Unwin, 1969.

[ОПП2019] Освітньо-професійна програма «Інформатика» за спеціальністю 122 "Компютерні науки" (ф-т КНК, КНУ імені Тараса Шевченка).– 2019.– Режим доступу: http://csc.knu.ua/media/filer_public/06/cd/06cdfecd-f915-4240-8238-39ae555f1d64/bac_122_inf_2019.pdf

[Bourbaki1968] Bourbaki, N. (1968). Theory of sets. Elements of mathematics. Paris: Hermann

[КМУ2016] Постанова Кабінету Міністрів України № 261 від 23 березня 2016 року. – 2016.– Режим доступу: https://zakon.rada.gov.ua/laws/show/261-2016-%D0%BF#Text

[Mendelson1966] Mendelson, E. (1966) Introduction to Mathematical Logic. 2nd Edition, D. Van Nostrand Co., Inc., Princeton; Toronto; New York; London.

[Proverbs] Activity proverbs. https://proverbicals.com/activity

[Filoni2022] Filoni, M.: "Man is action, not being" Hegel contra Heidegger in an unpublished essay by Kojève. Philosophical Inquiries, vol. 8, No. 2, pp. 203–208 (2020)

[Spirkin1983] Spirkin, A. (1983). Dialectical materialism. Moscow: Progress Publishers.– 343 pages

[PhilLog2006] Philosophy of Logic: An Anthology by Dale Jacquette (2006) Publisher: Wiley-Blackwell, 388 pages

[Harel1984] David Harel, Dynamic Logic, In D. Gabbay and F. Guenthner, editors, Handbook of Philosophical Logic, volume II: Extensions of Classical Logic, chapter 10, pages 497-604. Reidel, Dordrecht, 1984.

[Abrial1996] Abrial, J.-R.: Assigning Programs to Meanings, Cambridge University Press (1996)

[Spivey1992] Spivey, M.: The Z Notation: A Reference Manual, 2nd edition. Prentice Hall International Series in Computer Science (1992)

[RAISE1992] The RAISE specification language. Prentice Hall Int., 397 p. (1992)

[Lamport2002] Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers, Pearson Education Publ. (2002)

[Kleene1952] Kleene, S. C.: Introduction to Metamathematics, Van Nostrand, New York,1952

[NikSh2008] Nikitchenko M.S., Shkilnyak S.S., Mathematical logic and theory of algorithms, Publishing house of Taras Shevchenko National University of Kyiv, Kyiv, 2008, (in Ukrainian)

[NikSh2013] Nikitchenko, M., Shkilniak, S.: Applied logic. Kyiv: VPC Kyivskyi Universytet, in Ukrainian (2013)

[Kroening2008] Kroening D., Strichman O., Decision Procedures — an Algorithmic Point of View, Springer-Verlag, Berlin Heidelberg, 2008

[KrNik2013] Kryvolap, A., Nikitchenko, M., Schreiner, W.: Extending Floyd-Hoare logic for partial pre- and postconditions. In: Communications in Computer and Information Science, vol. 412, pp. 355–378. Springer, Cham (2013)

[NikMam2019] Nikitchenko, M., Shkilniak, O., Shkilniak S., Mamedov, T.: Completeness of the first-order logic of partial quasiary predicates with the complement composition. Computer Science Journal of Moldova 80 (2), 162-187 (2019)

[Marques-Silva2008] Marques-Silva J., Practical Applications of Boolean Satisfiability, In: Workshop on Discrete Event Systems (28-30 May 2008, Goteborg, Sweden), 74-80

[Nieuwenhuis2006] Nieuwenhuis R., Oliveras A., Tinelli C., Solving SAT and SAT modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T), J ACM., 2006, 53, 937-977

[Davis1952] Davis M., Logemann G., Loveland D., A machine program for theorem-proving, COMMUN ACM., 1962, 5(7), 394-397

[Janssen1997] Janssen T.M.V., Compositionality, In: van Benthem J., ter Meulen A. (Eds.), Handbook of Logic and Language, Elsevier and MIT Press, Amsterdam/Cambridge, 1997

[NikTym2012] Nikitchenko M.S., Tymofieiev V.G.: Satisfiability Problem in Composition-Nominative Logics of Quantifier-Equational Level. In: Proc. 8-th Int. Conf. ICTERI 2012, Kherson, Ukraine, June 6-10, 2012. CEUR-WS.org/Vol-848, http://ceur-ws.org/Vol-848/ICTERI-2012-CEUR-WS-paper-38-p-56-70.pdf (2012)

[Abramsky1993] S. Abramsky, D. Gabbay, T. Maibaum (editors). Handbook of Logic in Computer Science, Oxford University Press, V**ol. 1–5**, (1993–**2000).**

[Nik2001] N.S. Nikitchenko, Applicative Compositions of Partial Predicates. Cybernetics and Systems Analysis, vol. 37, issue 2 (2001), pp. 161–174. https://doi.org/10.1023/A:1016786600489

[NikSh2017] M. Nikitchenko, S. Shkilniak. Algebras and logics of partial quasiary predicates, Algebra and Discrete Mathematics, vol. 23, No 2 (2017), pp. 263–278. http://admjournal.luguniv.edu.ua/index.php/adm/article/view/440

[NikSh2019] M. Nikitchenko, O. Shkilniak, S. Shkilniak. Program Logics of Renominative Level with the Composition of Predicate Complement, in Proceedings of the 15th International Conference on ICT. CEUR Workshop Proc., (accepted for publication) (2019).

[NikIv2010] Nikitchenko M. Programming with nominative data / M. Nikitchenko, Ie. Ivanov // Proc. of CSE2010 Int. Scientific Conference on Computer Science and Engineering, 20–22 September 2010. — Kosice, Slovakia. — P. 30–39.

[SkobNikIv2014] Skobelev V. G. On Algebraic properties of nominative data and functions / V. G. Skobelev, M. Nikitchenko, Ie. Ivanov // Communications in Computer and Information Science (CCIS). — Springer, 2014. — Vol. 469. — P. 117–138.

[NikIv2010a] Nikitchenko M. S. Stability and monotony of programs regarding structural transformations of data / M. S. Nikitchenko, Ie. V. Ivanov / Problems in programming. — 2010. — Issue 2–3. — P. 58–67. (in Russian)

[Hem2022] Hemmendinger, D.: Encyclopedia of Computer Science: Syntax, semantics, and pragmatics, pp. 1737–1738. https://dl.acm.org/doi/10.5555/1074100.1074848. Accessed February 22, 2022.

[Bjor2009] Bjørner, D.: Domain Engineering: Technology Management, Research and Engineering, JAIST Press, 507 p. (2009)

[Nik1998] Nikitchenko, N.S.: A composition-nominative approach to program semantics. Technical report, IT-TR 1998–020, Technical University of Denmark (1998)

[Scott1971] Scott, D., Steachy, C.: Toward a mathematical semantics for computer languages Oxford Programming Research Group Technical Monograph. PRG-6 (1971)

[Wern2012] Werning, M., Hinzen, W., Machery, E. (Eds.): The Oxford handbook of compositionality. Oxford University Press (2012)

[Tur2018] Raymond Turner, R.: Computational Artifacts. Springer. Towards a Philosophy of Computer Science, Springer, Berlin, Heidelberg (2018) https://link.springer.com/book/10.1007/978-3-662-55565-1

[Zhang2021] Zhang, W., Sun, Y., Oliveira, B.: Compositional Programming. ACM Trans. Program. Lang. Syst. 43, 3, Article 9 (August 2021), 61 pages (2021) https://doi.org/10.1145/3460228

[Bove2012] Bove, A., Krauss, A., Sozeau, M.: Partiality and Recursion in Interactive Theorem Provers - An Overview. Mathematical Structures in Computer Science, Cambridge University Press (CUP), ffhal-00691459f (2012)

[NikTym2012a] Nikitchenko, M., Tymofieiev, V.: Composition-Nominative Logics in Rigorous Development of Software Systems, UNISCON 2012, pp. 140-151 (2012)

[Niel1992] Nielson H.R., Nielson F.: Semantics with Applications: A Formal Introduction. John Wiley & Sons Inc (1992)

[Wins1993] Winskel G.: The Formal Semantics of Programming Languages. MIT Press, Cambridge (1993)

[Ack1989] Ackoff, R. L.: From data to wisdom. Journal of Applied Systems Analysis 15, pp. 3–9 (1989)

[Zins2007] Zins, C.: Conceptual approaches for defining data, information and knowledge. Journal of the American Society for Information science and Technology, 58, 4, pp. 479–493 (2007)

[Hey2004] Hey, J.: The Data, Information, Knowledge, Wisdom Chain: The Metaphorical link, published at Intergovernmental Oceanographic Commission, 2004, 17 pages. Available at http://web.archive.org/web/20070206032947/ioc.unesco.org/o

ceanteach-
er/OceanTeacher2/02_InfTchSciCmm/DIKWchain.pdf

[Nik2001] Nikitchenko, N.S.: Intensional aspects of the notion of program. Problems of Programming, Kiev, 3–4, pp. 5-13 (2001). In Russian.

[App2009] Apostoli, P., Hinnion, R., Kanda, A, Libert, T.: Alternative Set Theories. In: Philosophy of Mathematics, Irvine A.D. (ed.), pp. 461-491, Elsevier (2009)

[Nik2001n] Nikitchenko, N.S.: Abstract Computability of Nondeterministic Programs over Various Data Structures. In: Perspectives of System Information science. LNCS, vol. 2244, pp. 471–484. Berlin: Springer (2001)

[Bac1978] Backus, J.: Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. Communs. ACM, 21, pp. 613-641(1978)