MIKLÓS FERENCZI – MIKLÓS SZŐTS

MATHEMATICAL LOGIC FOR APPLICATIONS

2011

This book is recommended for those readers who have completed some introductory course in Logic. It can be used from the level MSc. It is recommended also to specialists who wish to apply Logic: software engineers, computer scientists, physicists, mathematicians, philosophers, linguists, etc. Our aim is to give a survey of Logic, from the abstract level to the applications, with an emphasis on the latter one. An extensive list of references is attached. As regards problems or proofs, for the lack of space, we refer the reader to the literature, in general. We do not go into the details of those areas of Logic which are bordering with some other discipline, e.g., formal languages, algorithm theory, database theory, logic design, artificial intelligence, etc. We hope that the book helps the reader to get a comprehensive impression on Logic and guide him or her towards selecting some specialization.

Key words and phrases: Mathematical logic, Symbolic logic, Formal languages, Model theory, Proof theory, Non-classical logics, Algebraic logic, Logic programming, Complexity theory, Knowledge based systems, Authomated theorem proving, Logic in computer science, Program verification and specification.

Acknowledgement of support:
Prepared within the framework of the project "Scientific training (matemathics and physics) in technical and information science higher education" Grant No. TÁMOP- 4.1.2-08/2/A/KMR-2009-0028.

Prepared under the editorship of Budapest University of Technology and Economics, Mathematical Institute.

Referee:
Károly Varasdi

Prepared for electronic publication by:
Ágota Busai

Title page design:
Gergely László Csépány, Norbert Tóth

ISBN: 978-963-279-460-0

Contents

Chapter 0

INTRODUCTION

1. Logic as an applied science. The study of logic as a part of philosophy has been in existence since the earliest days of scientific thinking. Logic (or mathematical logic, from now *logic*) was developed in the 19th century by Gottlob Frege. Logic has been a device to research foundations of mathematics (based on results of Hilbert, Gödel, Church, Tarski), and main areas of Logic became full-fledged branches of Mathematics (model theory, proof theory, etc.). The elaboration of mathematical logic was an important part of the process called "revolution of mathematics" (at the beginning of the 20th century). Logic had an important effect on mathematics in the 20th century, for example, on algebraic logic, non-standard analysis, complexity theory, set theory.

The general view of logic has changed significantly over the last 40 years or so. The advent of computers has led to very important real-word applications. To formalize a problem, to draw conclusions formally, to use *formal methods* have been important tasks. Logic started playing an important role in software engineering, programming, artificial intelligence (knowledge representation), database theory, linguistics, etc. Logic has become an interdisciplinary language of computer science.

As with such applications, this has in turn led to extensive new areas of logic, e.g. logic programming, special non-classical logics, as temporal logic, or dynamic logic. Algorithms have been of great importance in logic. Logic has come to occupy a central position in the repertory of *technical knowledge*, and various types of logic started playing a key roles in the modelling of reasoning and in other special fields from law to medicine. All these developments assign a place to Applied Logic within the system of science as firm as that of applied mathematics.

As an example for comparing the applications and developing theoretical foundations of logic let us see the case of artificial intelligence (AI for short).

AI is an attempt to model human thought processes computationally. Many non-classical logics (such as temporal, dynamic, arrow logics) are investigated nowadays intensively because of their possible applications in AI. But many among these logics had been researched by mathematicians, philosophers and linguists before the appearance of AI only from a theoretical viewpoint and the results were applied in AI later (besides, new logics were also developed to meet the needs of AI). In many respects the tasks of the mathematician and the AI worker are quite similar. They are both concerned with the formalization of

certain aspects of reasoning needed in everyday practice. Philosopher, mathematician and engineers all use the same logical techniques, i.e., formal languages, structures, proof systems, classical and non-classical logics, the difference between their approaches residing in where exactly they put the emphasis when applying the essentially same methods.

2. Classical and non-classical logics. Chapter 2 is devoted to "classical first-order logic" and to logics closely related to it, called "*classical logics*". Classical first-order logic serves as a base for every logic, therefore it is considered as the most important logic. Its expressive power is quite strong (contrary to propositional logic, for example) and it has many nice properties, e.g. "completeness", "compactness", etc., (in contrast to second-order logic, for example). It is said to be the "logic of mathematics", and its language is said to be the "language of mathematics". The reader is advised to understand the basic concepts of logic by studying classical first-order logic to prepare the study of other areas of logic.

However, classical logics describe only static aspects of the modelled segment of the world. To develop a more comprehensive logical model multiple modalities are to be taken into consideration: - what is necessary and what is occasional, – what is known and what is believed, – what is obligatory and what is permitted, – past, present, future, – sources of information and their reliability, – uncertainty and incompleteness of information – among others.

A wide variety of logics have been developed and put to use to model the aspects mentioned above (in artificial intelligence, computer science, linguistics, etc.). Such logics are called *non-classical logics.* We sketch some important ones among them in Chapter 3 without presenting the whole spectrum of these logics, which would be far beyond the scope of this book.

3. On the concept of logic. Since many kinds of special logics are used in applications, a "general frame" has been defined for logic (see Chapter 1), which is general but efficient enough to include many special logics and to preserve most of their nice properties.

It is worth understanding logic at this general level for a couple of reasons. First, we need to distinguish the special and general features of the respective concrete logics anyway. Second, it often happens that researchers have to form their own logical model for a situation in real life. In this case they can specialize a general logic in a way suitable for the situation in question. *The general theory of logic* or *Universal Algebraic Logic* is a new, and quickly developing area inside logic (see Andréka, H., Németi, I., Sain, I., Universal Algebraic Logic, Springer, [14]).

Chapters 1. and 3. are based, among others, on the papers, Andréka, H., Németi, I., General algebraic logic: A perspective on What is logic, in What is logical system, Oxford, Ed. D. Gabbay, 1994, [11], and Andréka, H., Németi, I., Sain, I., Kurucz, A., Applying Algebraic Logic; A General Methodology. Lecture Notes of the Summer School "Algebraic Logic and the Methodology of Applying it", Budapest 1994, 67 pages, [13].

We note that there is a clear difference between a *concrete* logic (with fixed "non-logical symbols") and a *class* of concrete logics (only the "logical symbols" are fixed). The latter is a kind of generalization of the concrete ones, of course. Usually, by "logic" we understand a "class of logics", but the reader should be careful, the term "logic" because the term is used also for a concrete logic. We must not confuse the different degrees of generalizations.

4. Areas of mathematics connected with logic. An important aspect of this study is the connection between Logic and the other areas of mathematics. There are areas of mathematics which are traditionally close to Logic. Such areas are: algebra, set theory, algorithm theory.

For example, modern logic was defined originally in algebraic form (by Boole, De Morgan and Peirce). An efficient method in Algebra (in Logic) for problem solving is the following: translate the problem to Logic (to Algebra) and solve it in logical (in algebraic) form. The scientific framework of this kind of activity is the discipline called Algebraic Logic founded in the middle of the 20th century (by Tarski, Henkin, Sikorski, etc.). This area is treated in Chapter 4.

There are areas in mathematics which originally seemed fairly remote from Logic but later important and surprising logical connections were discovered between them. For example, such an area is Analysis. In the sixties, Abraham Robinson worked out the exact interpretation of infinitesimals through a surprising application of the Compactness Theorem of First Order Logic. Many experts believe this theory to be a more natural model for differential and integral calculus than the traditional model, the more traditional $\varepsilon - \delta$ method (besides analysis Robinson's idea was applied to other areas of Mathematics too, and this is called non-standard mathematics). This connection is discussed in Section 2.4.3.

We also sketch some connections between Logic and Probability theory (3.6.1).

5. The two levels of logics. Every logic has two important "levels": the level of *semantics* and that of *proof theory* (or proof systems or syntax). For most logics these two levels (two approaches) are equivalent, in a sense. It is important to notice that both levels use the same *formal language as a prerequisite*. So every logic has three basic components: *formal language, semantics and proof theory* (see [11], [13]). We make some notices on these components, respectively.

The concept of *language* is of great importance in any area of logics. When we model a situation in real life the first thing we choose is a suitable language more or less describing the situation in question. We note that today the theory of formal languages is an extensive, complex discipline and only a part of this theory is used in Logic directly.

Logical *semantics* is the part of logic which is essentially based on the theory of infinite sets. In general, in the definitions of semantics, there are no algorithms. Nevertheless, it is extraordinarily important in many areas of applications. Semantics is a direct approach to the physical reality.

Proof theory is the part of logic which is built on certain formal manipulations of given symbols. It is a generalization of a classical axiomatic method. The central concept of proof theory is the concept *of a proof system (or calculus)*. Setting out from proof systems algorithms can be developed for searching proofs. These algorithms can then be implemented on computers.

What is about the connection between these two approaches to Logic? The "strength" of a logical system depends on the degree of equivalence between the semantic and the proof-theoretical components of the logic (such result are grouped under the heading of "completeness theorems" for a particular logic).

The two levels of logic together are said to be the "double construction" of logic. First-order logic is complete, therefore, the "double construction" of logic

has repercussions with respect to the whole mathematics.

In addition to strength of a logic there are famous *limits* of logics (also that of first-order logic): *undecidability and incompleteness* (see Church and Gödel's results). These limits have interesting practical, philosophical and methodological consequences for the whole science.

Throughout the Chapters 1 and 2 (and partially in Chapter 3) we treat the main components of logic and their relationships systematically.

6. On the reading of the book. We suppose that the reader has some experience in Logic. This does not mean concrete prerequisites, but a kind of familiarity with this area. For example, the reader will understand Chapter 1, the general frame of Logic, more comprehensively if he/she knows concrete logics (in any case, the reader is urged to return once more to Chapter 1 after reading Chapters 2 and 3).

Today Logic is a large area inside science. To give a more or less comprehensive overview of this huge domain, we were forced to be selective on the topics and the theoretical approaches we survey. As regards the proofs of the theorems and further examples connected with the subject, we must refer the reader to the literature. Some important references are: [11], [13], [14], [34], [51], [145], [23], [43], [96], [156], [5], [71].

7. Acknowledgement. We say thanks to Gábor Sági for his useful advices and notices. Furthermore, we say thanks also to our students at the Technical University Budapest (to mathematicians and software engineers) for their valuable notes.

Chapter 1

ON THE CONCEPT OF LOGIC

In this chapter we give *general* definitions pertaining to classical and non-classical logics, which we *specialize, detail and illustrate with examples* later in the book. We present the general concepts concerning the main parts of logic: syntax, semantics, proof theory and their connection, respectively.

1.1 Syntax

First, we sketch the general concept of syntax of logics (the language $\mathcal{L}$). This kind of syntax does not differ essentially from the syntax used in symbol processing in Computer Science, or in a wider sense, musical notes in music or written choreography in dance.

Syntax is given in terms of a set of symbols called *alphabet* and a *set of syntactical rules*. In this book we assume that the alphabet is *countable* (finite or infinite), with possible exceptions being explicitly mentioned when necessary. The *alphabet* of a logic consists of two kinds of symbols. One of them are the *logical symbols*, the other are the *non-logical* ones. There is associated with each *non-logical symbol* particular a natural number or 0, the *arity* corresponding to the symbol. The sequences of such arities have importance in logic, it is called *the type* of the language.

A finite sequence of the alphabet which has a meaning (by definition) is called a formula. The set of formulas is denoted by $\mathcal{F}$ and is defined by the *syntactical rules*.

Besides formulas, there are other finite sequences of the alphabet which have importance, namely *terms*. These sequences are used to build up formulas. The term *expression* covers both formulas and terms.

Assume that in a language $\mathcal{L}$ there is given a set P of symbols (called *atomic formulas*) and another set Cn (called *connectives*) such that for every connective $c \in Cn$ has a natural number (the rank) k. Then, the set $\mathcal{F}$ of formulas coincide with the smallest set satisfying the conditions (i) and (ii) below:

(i) $P \subseteq \mathcal{F}$,

(ii) $c(\alpha_1, \dots \alpha_k) \in \mathcal{F}$,
where $\alpha_1, \dots \alpha_k$ are arbitrary formulas in $\mathcal{F}$, and the connective c has the rank k.

The terminology "*logical language*" (or "language", for short) is used *at least in two contexts* in the literature. The one is for a *concrete* logical language,the other for a *class of such concrete languages* (this latter is called *general* language). A *general language* is specified by the logical symbols used, while a *concrete language* is specified by the concrete non-logical symbols of the alphabet (for example, the operations and constants $+$, $\cdot$, $-$, 0, 1 as non-logical symbols specify the *concrete* first-order language of real numbers as a specialization of the *general* language of first-order logic).

Some remarks concerning syntax are:

- It is important to realize that the definition of a logical language, and also, almost the whole study of logic uses *metalanguage*s. The definitions in this book use *natural* language as the metalanguage, as it is usual.

- Generally, a logical language can be defined by *context-free formal grammar*: the alphabet and the types of the expressions correspond to the set of terminal symbols and to the non-terminal symbols, respectively.

  A terminological remark: *formulas* in a logical language correspond to "*sentences*" in a formal grammar. The word "*sentence*" in a logical language means a *special class of formulas*, which cannot be specified by a formal grammar. Let us think of programming languages, where programs can be defined by a context-free grammar, but important aspects of syntactic correctness cannot be described by thereby.

- Syntax can be defined as an *algebra* too: formulas form the universe of the algebra and the operations correspond to the rules of syntax. This algebra is a "*word algebra*" with equality being the same as identity (two different formulas cannot be equal). With the sets of atomic formulas and logical connectives in the language we can associate the word algebra *generated* by the set of atomic formulas using the given logical connectives as algebraic operations, in the usual algebraic sense.

- We can use *prefix (Polish notation), infix or postfix* notations for the expressions of the language. For example, taking a binary operation symbol O and applying it to the expressions α and β, the notations $O\alpha\beta$, $\alpha O\beta$ and $\alpha\beta O$ are prefix, infix and postfix notations, respectively. Any of these notational conventions has advantages as well as disadvantages. For example, the infix notation can be read easily, but *brackets and punctuation marks* (commas and points) are needed in the alphabet, and also, various precedence rules must be specified. Infix and postfix notations are useful to manipulate formulas by computers; e.g. to evaluate expressions automatically. For automated processing the so-called parsing tree provides an adequate representation of expressions.

There are two usual additional requirements concerning syntax:

- *The set of formulas should be a decidable subset composed from the alphabet (decidability of the syntax).*
- *Formulas should have the unique reading property.*

The unique reading property means that for every expression of the language there is only one way to construct it by the rules of syntax (that is, every expression has a unique parsing tree).

Most logical languages have both properties.

1.2 Basic concepts of semantics

First we introduce a *general concept of "logic"*, approaching this concept from the side of *logical semantics*.

Let us assume that a language $\mathcal{L}$ is given. Semantics is defined by a class of "*models*" (interpretations) and a "*meaning function*", which provides the meaning of an expression in a model. The following formal definition pertains to many well-known logics:

1.1 Definition (logic in the semantical sense) *A logic in the semantical sense is a triple*

$$L^S = \langle \mathcal{F},\ \mathcal{M},\ m \rangle$$

where $\mathcal{F}$ is a set of formulas in $\mathcal{L}$, $\mathcal{M}$ is a class (class of models or structures), and m is a function (meaning function) on $\mathcal{F} \times \mathcal{M}$, where we assume that the range of m is a partially ordered set.

Sometimes we shall denote the members of L^S in this way: $\langle \mathcal{F}_L,\ \mathcal{M}_L,\ m_L \rangle$.

1.2 Definition *The validity relation $\models$ ("truth for a formula on a model") is a relation defined on $\mathcal{F} \times \mathcal{M}$ in terms of the meaning function m (notation: $M \models \alpha$, where $\alpha \in \mathcal{F},\ M \in \mathcal{M}$) as follows:*

$$M \models \alpha \text{ if and only if } m(\beta, M) \leq m(\alpha, M) \text{ for every } \beta \in \mathcal{F} \tag{1.1}$$

where $\leq$ is the given partial ordering on the range of m and α is a fixed formula.

If there is a maximal element in the range of m, then (1.1) means that $M \models \alpha$ if and only if $m(\alpha, M)$ is maximal (for two-valued logic $M \models \alpha$ if and only if $m(\alpha, M)$ is true).

We note that it often happens that the validity relation is defined first, then the meaning function in terms of $\models$.

Now, we list some definitions concerning *logics* above:

1.3 Definition
A formula α is said to be universally valid *if $M \vDash \alpha$ for every model M, where $M \in \mathcal{M}$ (notation: $\vDash \alpha$).*

M is a model *of a given set Σ of formulas if $M \vDash \alpha$ for every $\alpha \in \Sigma$ (notation: $M \vDash \Sigma$).*

α is a semantical consequence *of a given set Σ of formulas if $M \vDash \alpha$ for every model M such that $M \vDash \Sigma$ (notation: $\Sigma \vDash \alpha$). The set of all semantical consequences of Σ is denoted by $Cons\Sigma$.*

A theory *of a given class $\mathcal{N}$ of models is the set Γ of formulas such that $\alpha \in \Gamma$ if and only if $M \vDash \alpha$ for every $M \in \mathcal{N}$ (notation: $Th\mathcal{N}$ or ThM if $\mathcal{N} = \{M\}$).*

A theory $Th\mathcal{N}$ *is* decidable *if the set $Th\mathcal{N}$ is a decidable subset of $\mathcal{F}$. In particular a* logic is decidable *if $Th\mathcal{M}$ is decidable, where $\mathcal{M}$ is the class of its models.*

If the truth values "true" and "false" are present in the range of the meaning function (they are denoted by t and f), then a formula α is called a sentence *if $m(\alpha, M) \in \{t, f\}$ for every model M.*

1.4 Definition
A logic has the compactness *property if the following is true for every fixed set Σ of formulas: if every finite set Σ' ($\Sigma' \subseteq \Sigma$) has a model, then Σ also has a model.*

A logic has *Deduction theorem* if there is a "formula scheme" $\Phi(\alpha, \beta)$ such that $\Sigma \cup \{\alpha\} \vDash \beta$ is equivalent to $\Sigma \vDash \Phi(\alpha, \beta)$ *(an important special case when $\Phi(\alpha, \beta)$ is $\alpha \to \beta$).*

Some comments on these definitions:

- *Models* (or *structures*) represent a particular domain of real life in logic (in a sense).
- Notice that the symbol $\vDash$ is used in three different senses (and the intended meaning is determined by the context):

  for validity relation: $M \vDash \alpha$,

  for universal validity: $\vDash \alpha$,

  for semantical consequence: $\Sigma \vDash \alpha$.
- Compactness is an important property of a logic because it allows a kind of finitization. An equivalent version of compactness is the following:

  If $\Sigma \vDash \alpha$, then $\Sigma' \vDash \alpha$ for some finite subset Σ' of Σ.
- Compactness and Deduction theorems together will provide a connection between the concepts of semantical consequence and universal validity.
- A version of the indirect inference rule is the following equivalence: $\Sigma \vDash \alpha$ if and only if $\Sigma \cup \{\neg\alpha\}$ has *no* model.
- Algebras can also be associated with models, but we skip the details here.

- Now we define another important concept of logic, that of *regular logic*. This is a stronger, but at the same time more specific concept than the concept of logic discussed above.

Let a language $\mathcal{L}$ and a model M be fixed. A connective is said to be *derived* in $\mathcal{L}$ if it can be defined in terms of the basic logical connectives of $\mathcal{L}$.

1.5 DEFINITION (regular logic) *A logic L^S is called a regular logic, if the following properties (i), (ii), (iii) are satisfied:*

(i) *(principle of compositionality). Assume that in $\mathcal{L}$ with every logical connective c of rank k an operation C of rank k is associated in the range of m. Then,*

$$m(c(\alpha_1, \dots \alpha_k), M) = C(m(\alpha_1, M), \dots m(\alpha_k, M))$$

must hold for arbitrary formulas $\alpha_1, \dots \alpha_k$.

(ii) *Assume that ∇ is a binary derived connective in $\mathcal{L}$ and T is a derived constant (as special connective) with the meaning "true". Then,*

$$M \vDash \alpha \nabla \beta \text{ if and only if } m(\alpha, M) = m(\beta, M), \text{ and}$$
$$M \vDash T\, \nabla \beta \text{ if and only if } M \vDash \beta.$$

are required.

(iii) *(substitution property) Assume that $\mathcal{L}$ contains a set Q of atomic formulas. Then, for an arbitrary formula α, containing the atomic formulas $P_1, \dots P_n$*

$$\vDash \alpha(P_1, \dots P_n) \text{ implies } \vDash \alpha(P_1/\beta_1, \dots P_n/\beta_n)$$

must hold for arbitrary formulas $\beta_1, \dots \beta_n$, where $P_1/\beta_1, \dots P_n/\beta_n$ denote the result of every occurrence of P_i being substituted simultaneously with β_i, $i = 1, \dots, n$.

(i) means that m "preserves" syntactical operations, that is, from the algebraic viewpoint, m is a *homomorphism* from the word algebra of formulas to the algebra corresponding to the model. Compositionality ensures that the meanings of formulas in a fixed model constitute such an algebra which is "similar" to the word algebra (this is one of the foundations of the so-called *algebraic semantics*).

In (ii), the operation ∇ is a weakening of the operation $\leftrightarrow$ (biconditional); therefore, if the biconditional can be formulated in the language, ∇ can be replaced in (ii) by $\leftrightarrow$.

For regular logics, it is possible to prove stronger (but still general) results than for those in Definition 1.1.

1.3 Basic concepts of proof theory

First, we define a central concept of proof theory: the concept of *proof system (or calculus)*.

A *proof system* is defined by a set of *axioms*, a set of *inference rules* and the concept of *proof*. Now we sketch these concepts, respectively.

Let us extend the language by an infinite sequence $X_1, X_2, \dots$ of new variables (called *formula variables*). First, we define the concept of "*formula scheme*" by recursion.

Formula schemes are obtained by applying finitely many times the following rules:

(i) the formula variables $X_1, X_2, \dots$ are formula schemes,

(ii) if $\Phi_1, \Phi_2, \dots$ are formula schemes and c is a k-ary logical connective in the language, then $c(\Phi_1, \Phi_2, \dots, \Phi_k)$ is also a scheme.

A formula α is an *instance of a scheme* Φ if α is obtained by substituting all the formula variables in Φ by given formulas.

An "*axiom*" of a calculus (a logical axiom) *is given* as a *formula scheme* (but the term "axiom" is usually used for both the scheme and its instance).

An *inference rule* is $\langle\langle \Phi_1, \Phi_2, \dots, \Phi_n\rangle, \Phi\rangle$, where $\Phi_1, \Phi_2, \dots, \Phi_n, \Phi$ are formula schemes, $\Phi_1, \Phi_2, \dots, \Phi_n$ are called the premises, Φ is called the conclusion. Another well-known notation for an inference rule is: $\frac{\Phi_1, \Phi_2, \dots \Phi_n}{\Phi}$.

The next important component of proof systems is the concept of *proof*. There are several variants of this concept. We define an important one together with the concept of *provability* for the case of the so-called *Hilbert style proof systems*. This definition is very general and simple.

Let us assume that a set of axioms and a set of inference rules are fixed (we skip the details).

1.6 Definition *A formula α is* provable (derivable) *from a set Σ of formulas (notation: $\Sigma \vdash \alpha$) if there is a finite sequence $\varphi_1, \varphi_2, \dots, \varphi_n$ (the* proof *for α) such that $\varphi_n = \alpha$ and for every $i \in n$,*

(i) *$\varphi_i \in \Sigma$, or*

(ii) *φ_i is an instance of an axiom (scheme), or*

(iii) *there are indices $j_1, j_2, \dots, j_k < i$ and an inference rule $\langle\langle \Phi_1, \Phi_2, \dots, \Phi_k\rangle, \Phi\rangle$ in the system such that $\left\langle\left\langle \varphi_{j_1}, \varphi_{j_2}, \dots, \varphi_{j_k}\right\rangle, \varphi_i\right\rangle$ is an instance of this rule (i.e. the formulas $\varphi_{j_1}, \varphi_{j_2}, \dots, \varphi_{j_k}$, φ_i are instances of the schemes $\Phi_1, \Phi_2, \dots, \Phi_k$, Φ in this rule, respectively).*

There is an important additional requirement for proof systems: *the set of axioms and the set of inference rules should be* decidable.

The relation $\vdash$ is called *the provability relation*. $\vdash$ is a relation on $\mathcal{P}(\mathcal{F}) \times \mathcal{F}$ (where $\mathcal{P}(\mathcal{F})$ is the power set of $\mathcal{F}$). If the proof system (calculus) is denoted by C, then the provability relation corresponding to C is denoted by $\vdash^C$ (if misunderstanding is excluded we omit C from $\vdash^C$).

With different proof systems C_1 and C_2 we associate different provability relations $\vdash^{C_1}$ and $\vdash^{C_2}$, but it is possible that the relations $\vdash^{C_1}$ and $\vdash^{C_2}$ coincide (this is true for every well-known calculus of first-order logic, for example).

Notice that with the concept of a proof system and the set Σ of formulas above we can associate the classical *axiomatic method*.

We can classify proof systems according to the way they are put to use. From this viewpoint there are two kinds of proof systems: *deduction systems* and *refutation systems*. For a deduction system we set out from premises to get the conclusions (e.g. Hilbert systems, natural deduction). For refutation systems we apply a kind of *indirect* reasoning: the premises and the *negation* of the desired conclusion are *simultaneously* assumed and we are going to "force" a contradiction in a sense to get the conclusion (e.g. resolution, analytic tableaux).

Now we introduce the proof-theoretical concept of "logic". Assume that a fixed proof system C is given.

1.7 Definition (logic in proof-theoretical sense) *A* logic *is a pair*

$$L^P = \langle \mathcal{F}, \vdash^C \rangle$$

where $\mathcal{F}$ is the set of formulas in $\mathcal{L}$ and $\vdash^C$ is the provability relation specified by the proof system C.

Sometimes the dependence on L is denoted in the members of L^P in this way: $\langle \mathcal{F}_L, \vdash^C_L \rangle$.

We list some definitions for proof theory (we omit C from $\vdash^C$).

1.8 Definition
If $\Sigma \vdash \alpha$ holds, we say that α is a proof-theoretical consequence *of Σ. The set $\{\alpha : \Sigma \vdash \alpha, \alpha \in \mathcal{F}\}$ of all the* proof-theoretical consequences of *a fixed Σ is denoted by $Ded\ \Sigma$.*
A set Σ of formulas is a theory *if $\Sigma = Ded\ \Sigma$ (that is Σ is* closed *under the provability relation).*
A theory Σ is inconsistent *if both $\Sigma \vdash \neg\alpha$ and $\Sigma \vdash \alpha$ hold for some formula α. Otherwise, the theory Σ is said to be* consistent.
A theory Σ is complete *if $\alpha \in \Sigma$ or $\neg\alpha \in \Sigma$ holds for every formula α.*
A theory Σ is decidable *if Σ is a decidable subset of $\mathcal{F}$.*
A theory Σ is axiomatizable *if $\Sigma = Ded\ \Sigma'$ for some recursive subset Σ' of $\mathcal{F}$.*

Now we list the most important general properties of a provability relation:

(i) *inclusion (reflexivity)*, that is $\beta \in \Sigma$ implies $\Sigma \vdash \beta$,

(ii) *monotonicity*, that is $\Sigma \vdash \alpha$ implies $\Sigma \cup \{\beta\} \vdash \alpha$

(iii) *cut,* that is $\Sigma \vdash \varphi$ and $\Lambda \cup \{\varphi\} \vdash \alpha$ imply $\Sigma \cup \Lambda \vdash \alpha$, where α, β and φ are formulas, Σ and Λ are arbitrary sets of formulas.

Finally, some words on the concept of *automatic theorem proving* (see Section 2.3.5). A proof system does not provide a decision procedure, that is, the provability relation is not a decidable relation. A proof system only provides a *possibility* for a proof. An old dream in mathematics is to generate proofs automatically. This dream got closer to reality in the age of computers. Algorithms have to be constructed from calculi from which a derivation of the required theorem is performed. Historically, resolution calculus was considered as a base for automatic theorem proving. Since then, the devices of automatic theorem proving have been multiplied.

1.4 On the connection of semantics and proof theory

Now we turn to the *connection between the two levels of logic*, to the connection between semantics and proof theory.

Let us consider a logic in semantic form and in syntactical form *together*, with the same set $\mathcal{F}$ of formulas: in this way, we obtain a more comprehensive notion of logic.

1.9 Definition *A* logic *is the sequence*

$$L = \langle \mathcal{F},\ \mathcal{M},\ m,\ \vdash^C \rangle$$

where the members of the sequence are the same as in the Definitions 1.1 and 1.7.

To obtain stronger results (e.g., proving completeness), it is necessary to assume that the semantical part of L is a regular logic.

We list some concepts concerning the connection between the consequence relation $\models$ and a provability relation $\vdash^C$.

1.10 Definition
A proof system C (or the relation $\vdash^C$, or the logic L) is strongly complete *if $\Sigma \models \alpha$ implies $\Sigma \vdash^C \alpha$ for every set Σ of the formulas and every formula α. If $\Sigma \vdash^C \alpha$ implies $\Sigma \models \alpha$ for every Σ and α, then the proof system is said to be* strongly sound.

A proof system (or the relation $\vdash^C$, or the logic L) is weakly complete *if $\models \alpha$ implies $\vdash^C \alpha$ for every formula α. In the opposite case, that is, if $\vdash^C \alpha$ implies $\models \alpha$, then the proof system is said to be* weakly sound.

Completeness theorems, i.e., theorems stating completeness together with soundness for a given logic, are basic theorems of logics. Most of the important logics have a kind of completeness property.

The *strong completeness theorem* is:

$$\Sigma \vDash \alpha \text{ if and only if } \Sigma \vdash^{C} \alpha$$

i.e. the semantical and the syntactical concepts of logical consequence are equivalent ($Cons\Sigma$ and $Ded\ \Sigma$ coincide).

Remarks on completeness:

- The main idea of proof theory is to reproduce the semantical concept $\Sigma \vDash \alpha$ (or only the concept $\vDash \alpha$), using only *finite manipulations* with formulas and avoiding the in-depth use of the theory of infinite sets included in the definition of $\Sigma \vDash \alpha$. Strong completeness of a logic makes it possible for us to use such a short cut (weak completeness makes it available the reproduction for the case when $\Sigma = \emptyset$).
- Weak completeness and compactness imply strong completeness, as can be shown.
- Another important version of strong completeness is:

  A set Σ of formulas is consistent if and only if Σ has a model.

  This version is the base of the famous *model method* for proving the relative consistency of a system.
- *Refutation systems* impose a condition on a set Γ of formulas having *no* model. Using this condition and the fact that $\Sigma \vDash \alpha$ if and only if $\Gamma = \Sigma \cup \{\neg\alpha\}$ has *no* model, we can prove $\Sigma \vDash \alpha$.

The following problem is of central importance in logic and it is closely related to completeness and incompleteness:

Is it possible to generate in a recursive way the formulas of $Th\mathcal{K}$, where $\mathcal{K}$ is any fixed class of models, i.e., is there a recursive set Σ of formulas such that

$$Th\mathcal{K} = Ded\ \Sigma \tag{1.2}$$

There are two important special cases of (1.2), the cases when $\mathcal{K} = \mathcal{M}$ and $\mathcal{K} = \{M\}$, where M is a fixed model and $\mathcal{M}$ is the class of the possible models. If the logic has weak Completeness theorem, then in the case $\mathcal{K} = \mathcal{M}$ the answer is affirmative for the problem (1.2). But in the case $\mathcal{K} = \{M\}$, if $Th\mathcal{K}$ is strong enough (i.e. recursive relations can be defined in the theory), by Gödel Incompleteness Theorem, the formula *does not exist* in general, so the answer is *negative* for problem.

Setting out from a logic in the semantical sense (Definition 1.1) we can speak of the (weak or strong) completeness of a logic *without a concrete proof system*.

1.11 DEFINITION *A logic (in the semantical sense) is complete (weakly or strongly) if there is a proof system $\mathcal{C}$ and a provability relation $\vdash^C$ such that supplementing the logic by $\vdash^C$ the resulting logic is complete (weakly or strongly).*

References to Chapter 1 are, for example: [11], [13], [30], [96].

Chapter 2

CLASSICAL LOGICS

In this chapter we mainly deal with first-order logic. The other logics treated in this chapter are closely related to the first-order one. By investigating them one can attain a better understanding about the limitations and the advantages of first-order logic.

2.1 First-order logic

First-order logic (FOL) plays an exceptional role among logics. Any other classical logic either has less expressive power (e.g. propositional logic) or does not have the nice properties which first-order logic has. In a direct or an indirect way almost every logic *can be reduced* to first-order logic in a sense (this does not mean that other logics should be ignored). First-order logic is said to be "*the logic of classical mathematics*". Though mathematics also uses propositional logic, second-order logic, etc. to a certain extent, the applications of these logics can be somehow simulated by first-order logic. The *language* $\mathcal{L}$ of first-order logic can be considered also as a *collection of basic mathematical notations*. First-order logic is *applied* not only in mathematics but in almost every area of Science, where logic is applied at all.

2.1.1 Syntax

The alphabet U of a first-order language $\mathcal{L}$ contains the connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\forall$ and $\exists$ with ranks 1, 2, 2, 2, 2, 1 and 1, respectively, the equality symbol $=$, a sequence $x_1, x_2, \ldots$ of individuum variables as *logical* symbols; furthermore a sequence $f_1, f_2, \ldots$ of function symbols (including the individuum constants) and a sequence $P_1, P_2, \ldots$ of relation symbols (including the propositional constants) as *non-logical* symbols. The numbers of the arguments of the function symbols and those of relation symbols are given by two sequences corresponding to those of function and relation symbols (with members of the sequences being natural numbers or 0, the two sequences being separated by a semicolon ;).

The symbols $\neg$, $\wedge$, $\vee$, $\forall$, and $\exists$ correspond to the words "not", "and", "or", "for every", "for some", respectively. Defining a first-order language means to specify the concrete (finite or infinite) sequences of its function symbols and

relation symbols (and also the sequences of the number of their argument). This double sequence is the type of the language.

For example, the alphabet of the language $\mathcal{L}_R$ of ordered reals contains the non-logical constants $+$, $\cdot$, $-$, 0, 1 and $\leq$ (the signs of addition, multiplication, minus, zero, one, less than or equal, respectively). The type of $\mathcal{L}_R$ is $\langle 2, 2, 1, 1, 0, 0; 2\rangle$, where ; separates the arities of function symbols and that of relation symbols.

Two remarks concerning the alphabet of $\mathcal{L}$:

- Operations which can be expressed in terms of other operations can be *omitted* from the alphabet. For example, $\rightarrow$ can be expressed in terms of $\neg$ and $\vee$, therefore, $\rightarrow$ can be omitted. Sometimes extra symbols are introduced for the truth values ("true" and "false") into the logical language.
- In first-order languages individuum constants and propositional constants may form a separate category among non-logical symbols. Here we consider them *as function symbols* and *relation symbols* with 0 argument, respectively.

We define the *expressions* of first-order languages: *terms* and *formulas*.

2.1 Definition Terms *are obtained by finitely many applications of the following rules:*

(i) *the individuum variables and the individuum constants are terms;*

(ii) *if f is an $n-$ary function symbol and $t_1, \dots t_n$ are terms, then $ft_1, \dots t_n$ is also a term.*

2.2 Definition First-order formulas *are finite sequences over the alphabet of $\mathcal{L}$, obtained by finitely many applications of the following rules:*

(i) *if P is a $n-$ary relation and $t_1, \dots t_n$ are terms, then $Pt_1, \dots t_n$ is a formula,*

(ii) *$t_1 = t_2$ is a formula, where t_1, and t_2 are terms,*

(iii) *if α and β are formulas, then $\neg\alpha$, $\wedge\alpha\beta$, $\vee\alpha\beta$, $\rightarrow \alpha\beta$, $\leftrightarrow \alpha\beta$ are formulas,*

(iv) *if x is any individuum variable and α is a formula, then $\exists x\alpha$ and $\forall x\alpha$ are formulas.*

We note that *first-order languages have the properties* of *unique readability* and *decidability*.

2.3 Definition *Formulas occuring in the definition of a given formula are called the* subformulas *of the given formula. A* scope *of a quantifier $\exists$ or $\forall$ in the formula $\exists x\alpha$ or $\forall x\alpha$ is the subformula α. A given occurrence of the variable x in a formula is* bounded *if this occurrence is in the scope of some*

quantifier, in the opposite case this occurrence is said to be free*. A variable* x *is a* free *variable in a formula if it has a free occurence. A formula is said to be a* sentence *(to be a* closed *formula) if it has no free variable.*

The formulas in (i) and (ii) are called *atomic formulas*.

The concept of the substitution of a free variable by another variable can also be defined, but, we skip the details.

Definitions (2.2) and (2.3) and the examples above use *prefix* notations for the sake of brevity. But we use the usual *infix* notations when manipulating formulas by hand (assuming that the language is extended by precedence rules and brackets, well known from *mathematical practice*).

Examples *for prefix expressions* in the language $\mathcal{L}_R$ of reals:
terms: $+01$, $\cdot x1$, $\cdot x + y0$, $\ {}^{-1}y1$,
atomic formulas: $\leq \cdot xy1$, $\leq 0\ {}^{-1}x$, $= \cdot 0xx$,
sentences: $\forall x \exists y = \cdot xy1$, $\forall x = \cdot x00$, $\forall x \to \neg = x0 \to = \cdot x^{-1}x1$,
formulas with free variable: $\exists y = \cdot xy1$, $\to \neg = x0 \to = \cdot x^{-1}x1$.

the infix versions of expressions above:
terms: $0 + 1$, $x \cdot 1$, $x \cdot (y + 0)$, $y^{-1} - 1$,
atomic formulas: $x \cdot y \leq 1$, $0 \leq x^{-1}$, $0 \cdot x = x$,
sentences: $\forall x \exists y (x \cdot y = 1)$, $\forall x (x \cdot 0 = 0)$, $\forall x (\neg x = 0 \to x \cdot x^{-1} = 1)$,
formulas with free variable: $\exists y (x \cdot y = 1)$, $\neg x = 0 \to x \cdot x^{-1} = 1$.

2.1.2 Semantics

2.4 Definition *A* model *(or* structure*)* $\mathcal{A}$ *of type of* $\mathcal{L}$ *is a sequence*

$$\mathcal{A} = \langle V^{\mathcal{A}},\ P_1^{\mathcal{A}}, P_2^{\mathcal{A}}, \ldots,\ f_1^{\mathcal{A}},\ f_2^{\mathcal{A}}, \ldots \rangle \tag{2.1}$$

where $V^{\mathcal{A}}$ *is a non-empty set (the* universe *of* $\mathcal{A}$*),* $P_1^{\mathcal{A}}, P_2^{\mathcal{A}}, \ldots$ *are concrete relations on* $V^{\mathcal{A}}$ *associated with the relation symbols* $P_1, P_2, \ldots$ *with arities given in* $\mathcal{L}$*,* $f_1^{\mathcal{A}}, f_2^{\mathcal{A}}, \ldots$ *are concrete functions on* $V^{\mathcal{A}}$ *associated with the function symbols* $f_1,\ f_2, \ldots$ *with arities given in* $\mathcal{L}$*.*

Briefly put, a model is a set $V^{\mathcal{A}}$ equipped with certain relations (with functions and individuum constants, in particular) on $V^{\mathcal{A}}$. The *type* of a model (structure) is that of the language used. The superindices $\mathcal{A}'s$ in (2.1) are omitted if misunderstanding is excluded.

The interpretations of the function-, constant- and relation *symbols* in $\mathcal{L}$ are defined in (2.1). The interpretations of terms on A can be defined as usual. However, the interpretation of *variables* is *not determined* by the model since individuum variables are expected to denote any element of the universe. A possible interpretation of the variables can be considered as a sequence q_1, $q_2, \ldots, q_i, \ldots$ with members from the universe V, corresponding to the sequence $x_1, x_2 \ldots, x_i, \ldots$ of the individuum variables, respectively. Thus q_1, $q_2, \ldots, q_i, \ldots$

(q for short) is the function on the natural numbers such that $q_i \in V$ and $x_i^{\mathcal{A}} = q_i$, $i = 1, 2, \ldots$.

We are going to define the interpretation of a formula α on $\mathcal{A}$ with free variables. It will be defined as *all the interpretations of the individuum variables* under which α is "true" on $\mathcal{A}$. This set of interpretations of the individuum variables will be called the "*truth set* of α in $\mathcal{A}$" and it is defined by formula recursion as follows:

2.5 Definition

(i) *If* $Pt_1, \ldots t_n$ *is an atomic formula, including the individuum variables* $x_{i_1}, \ldots x_{i_k}$, *where* P *is* P_1 *or* P_2 *or* P_3 *etc., then the truth set* $[Pt_1, \ldots t_n]$ *of* $Pt_1, \ldots t_n$ *in* $\mathcal{A}$ *is the set*

$$\left\{q \mid \left\langle t_1^{\mathcal{A}}(q), \ldots, t_n^{\mathcal{A}}(q)\right\rangle \in P^{\mathcal{A}}\right\} \tag{2.2}$$

in particular the truth set $[P(x_{i_1}, \ldots x_{i_k})]$ *is*

$$\left\{q \mid \langle q_{i_1} \ldots q_{i_n}\rangle \in P^{\mathcal{A}}\right\}. \tag{2.3}$$

Also, if $Pt_1, \ldots t_n$ *is* $t_1 = t_2$ *, then* $[t_1 = t_2]$ *is*

$$\{q \mid \ t_1(q) = t_2(q)\}.$$

(ii) *If* $[\alpha]$ *and* $[\beta]$ *are defined, then let the truth sets* $[\neg\alpha]$, $[\alpha \wedge \beta]$, $[\alpha \vee \beta]$, $[\alpha \to \beta]$, $[\alpha \leftrightarrow \beta]$ *be* $V^\omega \sim [\alpha]$, $[\alpha] \cap [\beta]$, $[\alpha] \cup [\beta]$, $[\neg\alpha] \cup [\beta]$, $[\neg\alpha] \cap [\neg\beta] \cup [\alpha] \cap [\beta]$, *respectively.*

(iii) *If* $[\alpha]$ *is defined, let*

$$[\exists x_i \alpha] = \left\{q \mid q_v^i \in [\alpha] \ \textit{for some } v \in V\right\}$$

and

$$[\forall x_i \alpha] = \left\{q \mid q_v^i \in [\alpha] \ \textit{for every } v \in V\right\}$$

where q_v^i *is obtained from* q *by substituting the* i*th member of* q_i *with* v.

We define the concepts of the meaning function and validity relation in terms of the concept of truth set:

2.6 Definition *The value of the* meaning function m *for the formula* α *and model* $\mathcal{A}$ *is the truth set* $[\alpha]$ *of* α *in* $\mathcal{A}$ *where the partial ordering on* V^ω *is set inclusion.*

It is easy to check that $\mathcal{A} \vDash \alpha$ if and only if $[\alpha] = V^\omega$. The truth values "true" and "false" are represented by V^ω and $\emptyset$.

In particular, if α is a sentence (a closed formula), α is true on $\mathcal{A}$ if and only if $[\alpha] = V^\omega$. α is called to be *true for an evaluation* q of the individuum variables (in notation $\mathcal{A} \vDash \alpha\,[q]$) if and only if $q \in [\alpha]$.

One of the main purposes in logic is to *find the "true propositions"* on a given model, that is to *find the theory* of the model.

For example, let $\mathcal{R} = \langle R, +^{\mathcal{R}}, \cdot^{\mathcal{R}}, 0^{\mathcal{R}}, 1^{\mathcal{R}} \leq^{\mathcal{R}} \rangle$ is the structure of the ordered real numbers, where R is the set of real numbers, and the operations, constants and the relation $\leq^{\mathcal{R}}$ are the usual ones. The type of $\mathcal{R}$ is the same as that of $\mathcal{L}_{\mathcal{R}}$. The theory $Th\mathcal{R}$ of the reals consists of all the true propositions on $\mathcal{R}$, formulated in terms of $\mathcal{L}_{\mathcal{R}}$.

Some comments on these definitions:

- To understand the intuition behind the concept of a truth set, see the following example. Let us consider the formula $x_3 \leq x_4$. The truth set of this formula is:

$$\{q \mid q \in V^{\omega},\ q_3 \leq' q_4,\ q_3, q_4 \in V\} \tag{2.4}$$

  where $\leq'$ is the interpretation of $\leq$.

  One can see that only the third and fourth members of the q's play a role in (2.4). Therefore, we can rewrite (2.4) in this way:

$$\{\langle q_3, q_4\rangle \mid q_3 \leq q_4,\ q_3, q_4 \in V\}.$$

  In general, since every formula has only finitely many free individuum variables, a truth set "depends on only finitely many members of the sequence of the individuum variables". Nevertheless, for the sake of uniformity, we assume that *formally* the relations corresponding to *formulas*, i.e. the truth sets are *infinite dimensional*. From a geometrical point of view this means that a truth set corresponding to a formula with n free variables can be seen as an infinite dimensional "*cylinder set*" based to an n-dimensional set.

- Notice that "the truth" of a proposition is encoded in our Definition(2.5) as the relation $\in$, in a sense: $\mathcal{A} \vDash \alpha\,[q]$ if and only if $q \in [\alpha]$. This corresponds to the general definition of relations: a relation R is called "true" in a point q if and only if $q \in R$.

- We note that there is another, more traditional way to define the relation $\mathcal{A} \vDash \alpha$, without the concept of truth sets.

- Notice that (iii) in Definition (2.5) reflects exactly the intended meaning of quantifiers "for every element of the universe" or "for some elements of the universe". Infinite unions or intersections also can be used to define quantifiers.

- We treat first-order logic "*with* equality". This means that equality $=$ is introduced in the language as a *logical symbol* and by definition, it denotes the identity relation in every model.

- Notice that if the quantifiers were omitted from the language, the *expressive power of the logic would be much weaker* (we would obtain a kind of propositional logic).

2.1.3 On proof systems and on the connection of semantics and proof theory

Some important positive results for first-order logic:

2.7 THEOREM *First-order logic is weakly complete and sound, so* $\models \alpha$ *if and only if* $\vdash \alpha$ *for some provability relation* $\vdash$.

In other words the theory of first-order logic is *axiomatizable*.

2.8 THEOREM *First-order logic is compact.*

A *consequence* of the theorems above is: *First-order logic is strongly complete*.

The so-called *Hilbert proof system*, for example, is a strongly complete proof system for first-order logic. But there are many other well-known strongly complete proof systems for first-order logic: deduction systems (natural deduction, Gentzen's sequent calculus, etc.) and refutation systems (resolution, analytic tableaux), too.

The axioms and inference rules of the Hilbert proof system:

2.9 DEFINITION Axioms *for first-order Hilbert system (Hilbert calculus)*

(i) $\alpha \to (\beta \to \alpha)$.

(ii) $(\alpha \to (\beta \to \delta)) \to ((\alpha \to \beta) \to (\alpha \to \delta))$.

(iii) $(\neg\alpha \to \neg\beta) \to (\beta \to \alpha)$.

(iv) $\forall x(\alpha \to \beta) \to (\alpha \to \forall x\beta)$, *where* x *is not free in* α.

(v) $\forall x\alpha(\dots x\dots) \to \alpha(\dots x/t\dots)$, *where* $\dots x\dots$ *denotes that* x *is a free variable in* α, *and* t *is such a term that the free variables occuring in* t *remain free after applying the substitution* x/t.

(vi) $x = x$,

(vii) $x = y \to t(\dots x\dots) = t(\dots x/y\dots)$.

(viii) $x = y \to \alpha(\dots x\dots) = \alpha(\dots x/y\dots)$.

The axioms (vi)-(viii) are called the *axioms of equality*.

2.10 DEFINITION The inference rules *are:*
$\langle\langle \Phi \to \Psi, \Phi\rangle, \Psi\rangle$ *(*modus ponens*),*
$\langle\langle \Phi\rangle, \forall x\Phi\rangle$ *(*generalization*).*

We must not confuse the relation $\models$ and the connective $\to$ but of course the deduction theorem establishes an important connection between them. The deduction theorem says that $\sigma \models \alpha$ if and only if $\models \sigma \to \alpha$, where σ and α are sentences. Among others, as such, the investigation of the consequence relation $\sigma \models \alpha$ can be reduced to the investigation of the special relation $\models \sigma \to \alpha$. This result can be generalized from σ to a finite set Σ of closed formulas.

There are famous *limitations* of first-order logic:

- First-order logic is *not decidable* (by Church theorem). This means that the *theory of the deducible sentences* (i.e. $Th\mathcal{M}$, where $\mathcal{M}$ is the class of first-order models) is *not decidable*.
- If a first-order *theory* is "strong" enough (i.e. recursive relations can be interpreted in the theory), then the theory is *not axiomatizable*, by Gödel's first Incompleteness Theorem; as a consequence, *not decidable* either (undecidability can be derived from Church theorem).

  Another version of incompleteness is the following: if an axiomatizable, consistent first-order theory is "strong" enough, then it is *not complete*.
- In general, a model M is not determined by its theory ThM (by Löwenheim–Skolem theorem, see Section 3.4.1).
- If a theory is strong enough, then its *inconsistency cannot be proven inside this theory* (by Gödel's second Incompleteness Theorem).

References to Section 2.1 are, for example: [51], [23], [25].

2.2 Logics related to first-order logic

In this section we are concerned with *propositional logic* as a *restriction*, and with second-order logic as an *extension* of first-order logic, respectively. Propositional logic is also called as 0th order logic (we do not discuss nth order logics, in general). Finally, we survey many-sorted logic which is a version of first-order logic (it is equivalent to that, in a sense).

2.2.1 Propositional Logic

Classical propositional logic is a *base* for every logic. It has many nice properties but its expressive power is not very strong. Nevertheless, there are many applications of this logic. It plays a central role in *logical design* (e.g. at designing electrical circuits) or at the foundations of *probability theory* (see the concept of "algebras of events", Section 4.2), among others.

We introduce the language and the semantics of propositional logic *independently* of first-order logic.

The *alphabet* of the language contains the logical connectives $\neg$, $\wedge$, $\vee$, $\to$, $\leftrightarrow$ and the infinite sequence $P_1, P_2, \ldots$ of propositional symbols as non-logical symbols.

2.11 Definition Formulas *are obtained by finitely many applications of the rules below:*

(i) *the propositional constants are formulas,*

(ii) *if* α *and* β *are formulas, then* $\neg\alpha$, $\wedge\alpha\beta$, $\vee\alpha\beta$, $\to \alpha\beta$, $\leftrightarrow \alpha\beta$ *are also formulas.*

Let $\mathcal{P}$ denote the set of propositional constants.

2.12 Definition A model M *in propositional logic is a mapping from the set of propositional constants to a set of two elements* $\{t, f\}$ *of truth values, that is a mapping* $g_M : \mathcal{P} \to \{t, f\}$.

Obviously $\{t, f\}$ can be considered with the order $t < f$.

2.13 Definition *(meaning function* $m(\alpha, M)$ *for a fixed model* M, $m(\alpha)$ *for short)*

(i) *if* α *is a propositional constant, then* $m(\alpha) = g_M(\alpha)$

(ii) *if* $m(\alpha)$ *and* $m(\beta)$ *are defined, then*

$m(\neg\alpha) = t$ *if and only if* $m(\alpha) = f$,
$m(\alpha \wedge \beta) = t$ *if and only if* $m(\alpha) = t$ *and* $m(\beta) = t$,
$m(\alpha \vee \beta) = f$ *if and only if* $m(\alpha) = f$ *and* $m(\beta) = f$,
$m(\alpha \to \beta) = f$ *if and only if* $m(\alpha) = t$ *and* $m(\beta) = f$,
$m(\alpha \leftrightarrow \beta) = t$ *if and only if* $m(\alpha) = m(\beta)$.

Propositional logic is obviously a regular logic.

It is easy to check that the *validity relation* $\models$ (or *truth evaluation)* satisfies the following one: $M \models \alpha$ holds if and only if $m(\alpha) = t$ holds.

It is customary to define m by a truthtable. There are also other ways to define the meaning function, for example, as a special case of the meaning function defined in the first-order case or to introduce it as a homomorphism into *Kripke* models (see later in Section 3.1) or using the Boolean algebra of two elements (see in Section 4.1).

As regards *the proof theory* of propositional logic, all the proof systems mentioned in the first-order case have a version for propositional logic.

All the nice properties of first-order logic – strong completeness, compactness, deduction theorem, etc. – are true for propositional logic. Beyond these common properties there is an important *difference* between first-order logic and propositional logic: propositional logic is *decidable*. The reason of decidability is the finite character of the definition of truth evaluation (and that of the meaning function): in propositional logic there are no variables running over an infinite set (there are no quantifiers).

There is a close connection between propositional logic and the part of first-order logic which does not include variables and quantifiers. In this fragement terms are built from constants and function symbols, step by step. Such terms are called *ground terms*. Formulas are built as in first-order logic, but instead of terms, only ground terms are used in their definition. Formulas defined in this way are called *ground formulas*. Atomic ground formulas can be interpreted in first-order models. Since there are no variables, all of the ground formulas are true or false. The meaning function of a compound formula can be defined in the same way as for propositional logic. This latter kind of logic is widely used in resolution theory.

2.2.2 Second order Logic

The same way as first-order logic is an extension of propositional logic, second-order logic is an extension of first-order logic. In first-order languages, quantifiers apply *only* to individuum variables. If we need quantifiers to apply to relations or functions in a sense, we have to use second-order logic. We shall see, that the expressive power of second-order logic is stronger than that of first-order logic but the nice properties of first-order logic are *not* inherited.

We introduce second-order logic as an extension of the first-order one and only the differences will be mentioned.

The alphabet of first-order logic is extended by two new kinds of symbols, those of relation and function *variables*. The difference between relation constants and relation variables (function constants and function variables) is similar to that of individuum constants and individuum variables.

So the *alphabet* of second-order logic is such an extension of the first-order one which contains the following new logical symbols: for every natural number n a sequence of relation *variables* $X_1^n, X_2^n, \ldots$ and a sequence of function *variables* $U_1^n, U_2^n, \ldots$ with rank n (as well as the usual sequences of individuum variables, relation *constants* $P_1, P_2, \ldots$ and function *constants* $f_1, f_2, \ldots$ that are already part of the languages).

The definitions of second-order *terms* and second-order *formulas* are also extensions of those of first-order ones. The following additional rules are stipulated:

For terms:

- If $t_1, \ldots t_n$ are terms and U^n is a function *variable* with rank n, then $U^n t_1, \ldots t_n$ is a term.

For formulas:

- If $t_1, \ldots t_n$ are terms and X^n is a relation *variable* with rank n, then $X^n t_1, \ldots t_n$ is an atomic formula.
- If α is a formula and Y is any (individuum-, relation- or function-) variable, then $\forall Y \alpha$ and $\exists Y \alpha$ are also formulas.

For example, the formalization of the property "well-ordered set" is a second-order formula (an ordered set is well ordered if every non-empty subset has a minimal element):

$$\forall X(\exists z(z \in X \to \exists y(y \in X \wedge \forall z(z \in X \to y \leq z))))$$

where X denotes a unary relation variable.

As regards the *semantics* of second-order logic, the concepts of first-order model and *second-order model* coincide.

The definitions of the *meaning function* and the validity relation (truth on a model) are analogous with the first-order case, only some additional conditions are needed.

Proof theory can also be defined for second-order logic, but we do not go into the details here.

Most of the nice properties of first-order logic fail to be true for second-order logic. For example, completeness and compactness fail to be true. Since there is no completeness, the role of proof theory is different from that of first-order logic, it is "not equivalent" to the semantics.

Remember that we prefer decidable calculi. Undecidable inference rules can be defined for variants of second-order logic to make them complete.

Second order logic is much more weaker than first order logic, but its expressive power is considerable. Second-order logic occurs in mathematics and at applications. There are important properties (second-order properties) which can be formalized only by *second-order formulas*. Such properties are among others: "the scheme of induction", "well-ordered set", "Archimedian property", "Cantor property of intervals", etc. Second order properties of graphs are important in complexity theory. Sometimes, second-order formulas are replaced by infinitely many first-order ones. But the expressive power of the latter is weaker than that of second-order logic.

We list some important properties of classical logics in the following table:

logic	*Axiomatizability*	*Decidability*	*Compactness*
propositional	yes	yes	yes
1^{st} *order*	yes	no	yes
2^{nd} *order*	no	no	no

Here axiomatizability means that the relation $\models$ (semantic consequence) is recursively enumerable, decidability means that $\models$ is recursive (or equivalently, $Th\mathcal{M}$ is axiomatizable or decidable, where $\mathcal{M}$ is the class of first-order models).

n-th order logics ($n \geq 3$) are generalizations of second-order logic. Their logical status is similar to that of the second-order one. The approach in which all the n-th order variables ($n = 0, 1, 2, \ldots$) are considered simultaneously is called *type theory*. Type theory can be considered as "ω-order logic".

An extraordinary important result (due to Leon Henkin) is in higher order Logic that there is a semantics for the type theory such that the logic obtained in this way is complete. This semantics is sometimes called *internal* or *Henkin's semantics*, and the models of this semantics are called *weak models* or internal models. The idea of this semantics is that only those evaluations of the variables are defined in the model which are necessary for completeness. In this way, a complete calculus can be defined also for second order logic. Furthermore, this semantics applies, e.g., in non-standard analysis, algebraic logic, etc.

2.2.3 Many-sorted logic

Many-sorted logic is a version of first-order logic. Quantifiers and the arguments of the functions and relations are *restricted* in a sense. Many-sorted logic can be *applied* on the areas, where non-homogeneous structures occur. Such area is, for example, the theory of vector spaces. Vector space can be defined as a so-called "two-sorted" structure consisting of the set of *vectors* and the set of *scalars*. We can use many-sorted logic when we would have to use higher-order logics or non-standard analysis, among others. It is widely used in the theory of computer science, in defining abstract data types or in dealing with typed programming languages.

When defining many-sorted logic we set out from first-order logic. Let us fix a non-empty set I, the elements of which are called *sorts*.

2.14 Definition
The alphabet of many-sorted logic has
the usual logical symbols: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\forall$ *and* $\exists$, *further*
the individuum variables and the equality are "indexed" by values from I *(by the sorts), so for every* i $(i \in I)$ *we have*
a sequence $x_1^i, x_2^i, \ldots$ *of individuum variables,*
an equality symbol $=_i$.
There is a finite or infinite sequence $f_1, f_2, \ldots$ *of function symbols and a sequence* $P_1, P_2, \ldots$ *of relation symbols as in ordinary first-order logic.*
Further, with every relation symbol P *a finite sequence of sorts* $\langle i_1, \ldots i_n \rangle$ $(i_1, \ldots i_n \in I)$ *is associated, where* n *is the rank of the relation symbol. Similarly with every function symbol* f *a finite sequence of sorts* $\langle i_1, \ldots i_n, i_{n+1} \rangle$ $(i_1, \ldots i_n \in I)$ *is associated, where* n *is the rank of* f *and* i_{n+1} *is called the sort of the value of* f.

Terms of sorts are obtained by finitely many applications of the following rules:

(i) *the individuum variables and the constant symbols of their sorts are terms*,

(ii) *if the sorts of the terms* $t_1, \ldots t_n$ *are* $i_1, \ldots i_n$, *respectively, and* f *is a function symbol of sorts* $\langle i_1, \ldots i_n; i_{n+1} \rangle$, *then* $ft_1, \ldots t_n$ *is also a term of sort* i_{n+1}.

The definition of *formulas* is the usual recursive definition, ensuring consistency between sorts. Only the definitions of atomic formulas need to be modified somehow:
If R *is a relation symbol of sorts* $\langle i_1, \ldots i_n \rangle$ *and* $t_1, \ldots t_n$ *are terms of sorts* $i_1, \ldots i_n$ *respectively, then* $Rt_1 \ldots t_n$ *is a formula.*

2.15 Definition *A many-sorted model* M *is a sequence which contains*
a collection $\{V_i \mid i \in I\}$ *of sets (corresponding to the universe of a first-order model)*

a subset $P^M \subseteq V_{i_1} \times \ldots \times V_{i_n}$ *for every relation symbol* P *with sort* $i_1, \ldots i_n$ *in the language,*
a function $f^M : V_{i_1} \times \ldots \times V_{i_n} \to V_{i_{n+1}}$ *for every function symbol* f *with sorts* $i_1, \ldots i_n, i_{n+1}$ *in the language.*

The definitions of the *meaning function* m and the concept of validity relation are obvious (taking into account that the variable x_k^i runs on V_i).

Many-sorted logics can be transformed into first-order logic. Let us introduce a new relation symbol R_i for every sort i to perform the following translation: The translation of a sentence $\exists x^i \alpha$ is $\exists x(R_i(x) \wedge \alpha)$, and that of $\forall x^i \alpha$ is $\forall x(R_i(x) \to \alpha)$. The quantifiers in these formulas are often called *bounded quantifiers*.

Conversely, it is often simple to translate first-order formulas to many-sorted formulas.

For example, we often need first-order formulas of the form $\forall x(P_1 x \to \beta)$ or $\exists x(P_1 x \wedge \beta)$. They can be converted into many-sorted formulas $\forall x^1 \beta$ or $\exists x^1 \beta$ introducing the sorts indexed by 1 and 2 and the variables x^1 and x^2, where x^1 runs over the set of sort 1 that is over elements of the original universe with property P_1.

Because of the close connection with first-order logic, all properties of first-order logic remain true for many-sorted logic.

References to Section 2.2 are, for example: [11], [13], [51], [74], [120].

2.3 On proof theory of first order logic

Proof theory plays an important role in many applications of logic, e.g. in logic programming, in program specifications, etc. In this section we sketch two proof systems: the deduction system called *natural deduction* and the refutation system called *resolution*. This latter is the base of the PROLOG logic programming language and historically, it was the base of automatic theorem proving.

2.3.1 Natural deduction

The origin of this calculus goes back to [76], and we present it in the form as it is presented in [135]. This calculus reflects the construction of mathematical proofs.

First we treat the *inference rules* of the system. For any logical symbol there are inference rules of two types:

- *introduction rule*: this rule produces a formula from formulas in terms of logical symbol, like $\frac{\alpha \ \beta}{\alpha \wedge \beta}$;
- *elimination rule*: this rule dissects a formula eliminating a logical symbol, like $\frac{\alpha \wedge \beta}{\alpha}$.

$\wedge$I :	$\dfrac{\alpha \quad \beta}{\alpha \wedge \beta}$	$\wedge$E :	$\dfrac{\alpha \wedge \beta}{\alpha} \quad \dfrac{\alpha \wedge \beta}{\beta}$
$\vee$I :	$\dfrac{\alpha}{\alpha \vee \beta} \quad \dfrac{\beta}{\alpha \vee \beta}$	$\vee$E :	$\dfrac{\alpha \vee \beta \quad \begin{array}{c}[\alpha]\\ \Downarrow \\ \gamma\end{array} \quad \begin{array}{c}[\beta]\\ \Downarrow \\ \gamma\end{array}}{\gamma}$
$\rightarrow$ I	$\dfrac{\begin{array}{c}[a]\\ \Downarrow \\ \beta\end{array}}{\alpha \rightarrow \beta}$	$\rightarrow$ E	$\dfrac{\alpha \quad \alpha \rightarrow \beta}{\beta}$
$\neg$I	$\dfrac{\begin{array}{c}[\alpha]\\ \Downarrow \\ \bot\end{array}}{\neg\alpha}$	$\neg$E	$\dfrac{\begin{array}{c}[\neg\alpha]\\ \Downarrow \\ \bot\end{array}}{\alpha}$
$\bot$I	$\dfrac{\alpha \quad \neg\alpha}{\bot}$	$\bot$E	$\dfrac{\bot}{\alpha}$
$\forall$I :	$\dfrac{\beta(x_i/a_j)}{\forall x_i \beta}$	$\forall$E :	$\dfrac{\forall x_i \beta}{\beta(x_i/t)}$
$\exists$I	$\dfrac{\beta(x_i/t)}{\exists x_i \beta}$	$\exists$E :	$\dfrac{\exists x_i \beta \quad \begin{array}{c}\beta[x_i/a_j]\\ \Downarrow \\ \gamma\end{array}}{\gamma}$
$=$ I	$\dfrac{}{t = t}$	$=$ E	$\dfrac{t_1 = t_2 \quad \alpha}{\alpha(t_1/t_2)}$

Table 2.1: Rules of Natural Deduction

The following table includes the inference rules of Natural deduction. If $\mathcal{C}$ is a connective or a quantifier, the corresponding elimination and introduction rules are denoted by $\mathcal{C}$E and $\mathcal{C}$I respectively.

The syntactic vocabulary is expanded by the symbol $\bot$ to denote falsity, and by an enumerable set of constants $\{a_i\}_{i\in\omega}$. In the table t, t_1, t_2 are terms of the expanded language. In rule $=$E one is allowed to substitute only some occurrences of t_1 by t_2, in the other cases x_i/t means that all occurrences of x_i are to be substituted by t. In rules $\forall$I and $\exists$E the constant a_j is used with the condition that it occurs neither in the assumptions of the derivation of $\beta[x_i/a_j]$ or of the subderivation $\beta[x_i/a_j] \Rightarrow \gamma$ respectively, nor in the conclusion γ.

As examples we explain the rules $\vee$E and $\rightarrow$I from the table

The meaning of the $\vee$E rule is: If the premis is $\alpha \vee \beta$, and also a proof of γ from α, and a proof of γ from β are given, then we can conclude γ. The formalized version of this inference has also to include proofs of γ from α and β respectively. "The proof of γ from α" is denoted by $[\alpha] \Longrightarrow \gamma$ (the notation will occur vertically in the rules). The two proofs of γ depend on α and β respectively, however *neither* α *nor* β is a premis of the whole inference. So if the two "subproofs" gets into the proof of γ from the disjunction $\alpha \vee \beta$, assumptions α and β have to be discharged (cancelled). Brackets [] denote discharging.

Let us consider the of $\rightarrow$I rule. The elimination rule for implication is Modus

Ponens, but how to prove $\alpha \to \beta$? The meaning of the $\to$I rule is the following: if there is a derivation of β from α (i.e. $[\alpha] \Rightarrow \beta$) then $\alpha \to \beta$ can be considered to be proved. By Deduction Theorem this rule is expected to be correct. Clearly whereas α is a premise of the derivation of β, α is not a premise of $\alpha \to \beta$, so α has to be discharged.

If a rule includes the discharge of a formula, only *some occurrences* have to be discharged, maybe none of them (so discharge is always a possibility, but not an obligation).

The $\neg$E rule, the formal version of "reductio ad absurdum", is sometimes questioned in mathematics as well as in logics. If it is omitted, intuitionistic logic is obtained. This rule is equivalent to the "excluded middle" principle. If even the rule $\bot$E is omitted, then proof system of the so called minimal logic is obtained.

In Natural Deduction a *proof* of a formula α from a set of formulas Σ has a tree structure. The root of the tree is labeled by formula α, the elements of Σ may label some leaves, and some leaves may be labeled by a discharged formula. The other nodes can be constructed using the inference rules. We do not define here the structure of Natural Deduction proofs in an exact way, but two proofs are given as examples; namely the proofs of the universal validity of formulae $.\alpha \to \neg\neg\alpha$ and $\exists x_i \neg\alpha \to \neg\forall x_i \alpha$.

To understand the derivations the discharged formulae and the step when they are discharged are marked by the same index.

$$\dfrac{\dfrac{\dfrac{[\neg\alpha]^1 \quad [\alpha]^2}{\bot}\,\bot\text{I}}{\neg\neg\alpha}\,{}^1\,\neg\text{I}}{\alpha \to \neg\neg\alpha}\,{}^2\,\to\text{I}$$

In the second example elimination rules are also used. Notice that first the elimination rules are used, and after them the introduction rules.

$$\dfrac{\dfrac{\dfrac{\dfrac{[\exists x_i \neg\alpha]^3 \qquad \dfrac{[\neg\alpha(x_i/a)]^1 \qquad \dfrac{[\forall x_i \alpha]^2}{\alpha(x_i/a)}\,\forall\text{E}}{\bot}\,\bot\text{I}}{\bot}\,{}^1\exists\text{E}}{\neg\forall x_i \alpha}\,{}^2\neg\text{I}}{\exists x_i \neg\alpha \to \neg\forall x_i \alpha}\,{}^3\,\to\text{I}$$

The Natural Deduction proof system is *complete* for the semantical consequence relation of first order classical logic. If the $\neg$E rule is omitted, it is *complete* for the intuitionistic semantical consequence relation.

An important problem of proof theory is to establish the existence of the so called *normal form* of a proof, that is, the simplest derivation. Note that the inference rules have a kind of symmetry - the elimination and introduction rules can be considered as the inverses of each other. The application of an elimination rule to a formula obtained by its inverse introduction rules is superfluous. In a *normal form* such pairs are not allowed. More formally, each "path"[1] of a normal derivation consists of a (maybe empty) sequence of applications of introduction rules being followed by a (maybe empty) sequence of applications of elimination rules.

[1] The definition of the notion of path in a Natural Deduction proof is complicated, that is why we do not discuss it here.

The following proposition (the basic theorem of proof theory) holds: *Every derivation can be transformed into a normal form.*

2.3.2 Normal forms

To prepare the ground for the discussion of resolution we survey the normal forms of first order logic. Such "canonical" forms make the manipulation of formulae easier.

2.16 Definition (prenex normal form) *If a formula α is of the form $Q_1x_1 \dots Q_nx_n\alpha'$, where Q_i can be universal or existential quantifier, and α' is quantifier free formula, it is said to be in* prenex normal form.

The sequence of quantifications $Q_1x_1 \dots Q_nx_n$ is called the *prenex* (or *prefix*), and formula α' is called the *matrix* (or the *kernel*) of α.

For every first order formula there is an equivalent one in prenex normal form. We remark that prenex normal forms of a formula are not unique.

The next type of normal forms we consider is a simplification of the prenex normal form by eliminating existential quantifiers.

Notice that in a given model the truth of $\forall x_1 \exists y_1 \alpha$ means the existence of a function $Y_1(x)$ such that $\forall x_1 \alpha(y_1/Y_1(x_1))$ is true in this model. The underlying idea of universal *Skolem normal form* is that the language is expanded by function a function symbol to denote function Y_1. The function symbols so introduced so are called *Skolem functions.*

2.17 Definition (Skolem normal form) *Let α be a sentence in prenex normal form. The following algorithm generates the* Skolem normal form *of α:*

1. *If the prenex of α consists of only universal quantifiers, the algorithm stops, α is already in Skolem normal form.*
2. *Let α be of the form $\forall x_1 \dots \forall x_j \exists y_i \beta$ for some j and i. Let us expand the language by an j-ary function symbol Y_i which do not occur in the alphabet of the language, and let α be substituted by $\forall x_1 \dots \forall x_j \beta(y_i/Y_i(x_1, \dots x_j))$.*
3. *Go to 1., and repeat these steps.*

Notice that if $j = 0$ in step 2., that is α is $\exists y_1 \beta$, then the Skolem function turns out to be a constant function.

If in the definition the role of universal and existential quantifiers are reversed, the so called *existential Skolem normal form* is obtained.

Skolem normal form simplifies the prenex form, however equivalence to the original formula does not hold (they are only co-satisfiable):

2.18 Theorem (Skolem normal form) *The Skolem normal form of α is satisfiable if and only if α is satisfiable.*

The following definition provides a normal form for the matrix of a prenex formula.

2.19 DEFINITION (conjunctive normal form) *If a quantifier free formula β is of the form $(l_1^1 \vee \ldots \vee l_{n_1}^1) \wedge \ldots \wedge (l_1^m \vee \ldots \vee l_{n_m}^m)$, where l_i^j are literals (atomic formulae or negations of an atomic formulae), β is said to be in conjunctive normal form.*

The disjunctions $l_1^i \vee \ldots \vee l_{n_i}^i$ are called *clauses* (a finite disjunction of literals). A clause can be considered as the set of its literals.

Just like in the case of prenex normal form, for quantifier free formula there is an equivalent one in conjunctive normal form. Straightforward algorithms can be constructed to generate normal forms, but we do not discuss the details of them. We remark that the conjunctive normal forms of a formula are not unique. An appropriate concept of disjunctive normal form can also be defined, where the roles of $\vee$ and $\wedge$ are exchanged.

If the matrix of a formula in Skolem form is rewritten into conjunctive normal form, the so called strong Skolem form is obtained. Strong Skolem form syntheses all the other normal forms. The prefix can be moved into the matrix because of the conjunctive nature of the matrix:

$$\forall \overline{x} C_1(\overline{x}) \wedge \ldots \wedge \forall \overline{x} C_m(\overline{x}). \tag{2.5}$$

A strong Skolem form can be represented by the set $\{C_1, \ldots C_m\}$ of the clauses in its matrix. Here with a clause C we have to associate the universally quantified formula $\forall \overline{x}\ C$.

Notice that we can obviously achieve that the individuum variables in the respective clauses $C_1, \ldots C_m$ in 2.5 are disjoint sets. This procedure is called *separating the variables*.

2.3.3 Reducing the satisfiability of first order sentences to propositional ones

For the semantics of propositional logic there are effective ways to decide the consequence relation. Moreover it is quite easy to construct algorithms to compute it on different architectures. Although most practical problems require the application of *first order logic*, there do not exist effective algorithms for FOL (while there are such algorithms for propositional logic) This is why the so called Herbrand's theorem is of great importance. Herbrand's theorem reduces the unsatisfiability of a first order formula set to the unsatisfiability of a set of ground formulae, which can be considered propositional ones.

Let α be a quantifier free formula. A *ground instance* of α is a formula obtained by substituting variable free terms for the variables in α. The logic of ground instances is called *zero order logic*. Zero order logic is equivalent to propositional logic, since there is a one-to-one map between the set of the ground instances of atomic formulae and the set of the predicate letters.

Our goal is now is the following: for any first order formula set Σ, find a zero order one Σ_0 such that Σ is unsatisfiable if and only if Σ_0 is.

Suppose that language $\mathcal{L}$ does not include equality. Let α be of Skolem form $\forall x_1 \ldots \forall x_m \alpha$'. Let α_j^g denote a ground instance of α. A version of Herbrand's theorem is:

2.20 THEOREM (Herbrand's theorem) *$\forall x_1 \ldots \forall x_m \alpha'$ is unsatisfiable if and only if there are ground instances $\alpha_1'^g, \ldots, \alpha_n'^g$ of α' such that the set $\{\alpha_1'^g, \ldots, \alpha_n'^g\}$ is unsatisfiable.*

It is easy to check that there is an equivalent form of Herbrand's theorem concerning semantic consequence. Assume that the elements of Σ are of universal prenex forms and β is of existential prenex form. In this case $\Sigma \vDash \beta$ holds if and only if $\Sigma^g \vDash \beta_1^g \vee \ldots \vee \beta_m^g$ holds, where Σ^g includes the ground instances of the matrices of Σ, and β_i^g runs over all the ground instance of the matrix of β or every $(0 < i \leq m)$.

Herbrand's theorem can be used to extend propositional calculi to a first order ones - this is called "lifting" the calculus. The simplest one is outlined below: it has the property of completeness because of Herbrand theorem.

Let Σ be a set of quantifier free first order formulas and consider the set of ground instances Σ^g of Σ. To prove the unsatisfiability of Σ execute the following procedure:

2.21 DEFINITION (Davis–Putnam method) [2]

1. *Let Ψ be a finite subset of Σ^g.*
2. *Try to prove the unsatisfiability of Ψ by propositional logic.*
3. *If it is proved, let us terminate the procedure. If not, let the new Ψ a superset of Ψ in Σ^g and go to step 2.*

Since propositional calculi are decision calculi, the procedure terminates, if the sets of ground instances are choosen with care (e.g. ground instances of every formula in Σ are selected in some steps). The algorithm is non-deterministic: the selection of the and the ground substitutions in step 2 is not determined. A search strategy is, however, expected to define a deterministic algorithm.

Proof procedures based on Davis–Putnam method were applied in theorem provers (e.g. in resolution), but they proved to be hopelessly ineffective.

There is another, more effective method of lifting: *using metavariables*. If a human agent uses a calculus that is based on Herbrand's theorem, (s)he has a plan and maybe a good hypothesis what term has to be used when ground instances are generated. However a search strategy of a proof procedure is almost a blind search - that is why Davis–Putnam method is so ineffective. Let us recall the $\forall$E rule: $\frac{\forall x_i \beta}{\beta[t/x_i]}$. Here t is a metavariable standing for a term of the object language. If infinitely many ground terms are represented by a single *metavariable* in a proof procedure, then the procedure gets more effective. Later the process of finding the proof may create constraints for the metavariables, and the proper terms can be constructed. If this method is used, the syntactic

[2] This method was introduced for resolution calculus, however it can be used for any propositional calculus.

vocabulary needs to be supplemented by a set of metavariables. Lifting by metavariables can be applied any propositional calculus if it is applied to ground formulae.

2.3.4 Resolution calculus

Resolution is a relatively simple calculus, but the price of this simplicity is that the calculus can be applied directly to strong Skolem normal forms only. So strong Skolem normal form has to be constructed before starting resolution.

We noticed in the section on normal form that every formula can be represented by a finite set of clauses using the strong Skolem form of the formula. Moreover we shall see, during the resolution procedure only clauses are used, so the procedure resolution is, "closed" with respect to clauses. This is the background of the terminology: resolution uses the "language of clauses".

Resolution is a refutation calculus, where we apply a kind of indirect argument, that is, the set Σ of premises is supplemented by the negation of the formula α to be proven, and we aim at getting contradiction. Now we survey the preparation, needed to start the proof of $\Sigma \vDash \alpha$ using resolution calculus:

1 Form $\Sigma' = \Sigma \cup \{\neg\alpha\}$.

2 Convert the sentences in Σ' into strong Skolem normal forms.

3 Form the set of clauses occurring in these normal forms and separate the sets of the individuum variables occurring in them so that a set of clauses S is obtained.

Remember that a clause can be considered as a *set of literals*, therefore it is meaningful to speak about the clause consisting of one literal (called unit clause), and the empty clause (denoted by $\square$) too. The empty clause is *unsatisfiable*, since a clause is true in a model iff there is a literal in the clause that is true in the model. Therefore $\square$ can be considered as the symbol of *falsity*in the language.

An important notion connected with resolution is the complementary pair of literals: an atomic formula and its negation forms a *complementary pair*. It is not important which element of the pair is the atomic formula - therefore the notation l, $\neg l$ is introduced for complementary pair. $C \setminus \{l\}$ will denote the clause originated from the clause C omitting the literal l from the disjunction in C.

First we discuss resolution for propositional logic.

2.22 Definition (propositional resolution rule) *Clauses C_1, C_2 can be resolved if there are literals $l_1 \in C_1$, $l_2 \in C_2$ such that l_1, l_2 form a complementary pair*[3]*. In this case their resolvent is $Res(C_1, C_2)$ is $C_1 \setminus \{l_1\} \cup C_2 \setminus \{l_2\}$.*

$Res(C_1, C_2)$ can be denoted by $Res_{l_1,l_2}(C_1, C_2)$ too, and it is called a resolvent with respect to the complementary pair l_1 and l_2. C_1, C_2 are called the

[3]It is said that C_1, C_2 are clashing clauses or clashes on the complementary literals.

parents of resolvent $Res(C_1, C_2)$. Clearly the resolvent of a complementary pair is the empty clause.

It can be easily checked that $\{C_1, C_2\}$ is satisfiable if and only if $Res(C_1, C_2)$ is. This fact implies that the resolution rule alone constitutes a sound calculus.

2.23 DEFINITION (resolution proofs) *A clause C can be proved from a set S of clauses by propositional resolution if there is a finite sequence $C_1, C_2 \dots C_n = C$ of clauses such that for all $1 \leq i \leq n$ $C_i \in S$ or $C_i = Res(C_j, C_k)$ for some $j, k < i$.*

2.24 THEOREM (completeness of propositional resolution) *A set S of propositional clauses is unsatisfiable if and only if the empty clause $\square$ can be derived by propositional resolution from S.*

Resolution incorporates several well-known inference rules, as the following table shows.

parent clauses	resolvent	simulated rule	
$\{P\}, \{\neg P, Q\}$	$\{Q\}$	Modus Ponens:	$P, P \to Q \vdash Q$
$\{P\}, \{\neg P\}$	$\square$	$\neg$elimination:	$P, \neg P \vdash \bot$
$\{\neg P, Q\}, \{r, \neg Q\}$	$\{\neg P, r\}$	chaining:	$P \to Q,\ Q \to r \vdash P \to r$

Now let us turn to *first order resolution*. Resolution is define for the first order logic where $=$ is not a logical symbol, so the axioms of equality are not logical axioms.

The success of first order resolution is due to the first order resolution principle. A special way of lifting has been worked out for the resolution. It is known that if a pair of first order clauses has a pair of resolvable ground instances, usually there are infinitely many such pairs. The goal is to construct a clause as "first order resolvent" that can represent as MANY ground resolvents as possible - maybe all of them; as the scheme below shows:

$$\begin{array}{ccccc} C_1, C_2 & \longmapsto & \text{"first order resolution"} & \to & Res^1(C_1, C_2) \\ \downarrow & & & & \downarrow \\ \text{substitution} & & & & \text{substitution} \\ \downarrow & & & & \downarrow \\ C_1^g, C_2^g & \longmapsto & \text{"ground resolution"} & \to & Res^g(C_1, C_2) \end{array}$$

First some definitions connected with substitutions for the individuum variables have to be given. If p is an expression in $\mathcal{L}$ and σ is a substitution for some free variables, then σ applied to p will be denoted by $p\sigma$.

2.25 DEFINITION (unifier) *Given two expressions p_1, p_2, a unifier of them is a substitution σ that it makes them identical: $p_1\sigma = p_2\sigma$. If there is a unifier of p_1, p_2, they are said to be unifiable.*

A natural partial ordering between substitutions can be defined: σ_1 is *more general* than σ_2 if and only if there is a substitution λ such that $\lambda\sigma_1 = \sigma_2$.

2.26 Definition (most general unifier) *A substitution μ is called* most general unifier (mgu) *of expressions p_1, p_2 if and only if it is a unifier of p_1, p_2, and it is the most general among the unifiers of p_1, p_2, that is, for any unifier σ of p_1, p_2 there is a substitution λ such that $\lambda\mu = \sigma$. The most general unifier of p_1, p_2 is denoted by $mgu(p_1, p_2)$.*

2.27 Theorem *If two expressions are unifiable, they have a most general unifier.*

We emphasize that it is decidable whether two expressions are unifiable or not, moreover the most general unifier is computable.

Notice that deciding that two terms is unifiable means deciding that the two terms may denote the same element of Herbrand universe - that is, they have equal instances too. This fact is the basis of the following definition:

2.28 Definition *Literals a_1 and $\neg a_2$ constitute a* first order complementary pair *if and only if a_1 and a_2 are unifiable.*

2.29 Definition (first order resolution) *Clauses C_1, C_2 can be resolved if and only if there are literals $l_1 \in C_1$, $l_2 \in C_2$ such that l_1, l_2 form a first order complementary pair. Let $\mu = mgu(l_1, \neg l_2)$. Then $Res(C_1, C_2)$ is $(C_1\mu \setminus \{l_1\}\mu) \cup (C_2\mu \setminus \{l_2\}\mu)$.*

The first order resolution rule alone does not constitute a complete calculus, as the following example shows. Let us consider the clause set

$$\{\{r(x_1, x_2), r(y_1, y_2)\}, \{\neg r(x_3, x_4), \neg r(y_3, y_4)\}\}$$

which is clearly unsatisfiable. However, first order resolution generates only the variants of the clause $\{r(x_5, x_6), \neg r(y_5, y_6)\}$ and of the original clauses. Therefore first order resolution has to completed by a new rule.

2.30 Definition (factorization) *A clause can be factorized if there are two unifiable literals l_1, l_2 in C. Then a factor of C is $C\mu$, where $\mu = mgu(l_1, l_2)$.*

$C\mu$ can be denoted by C_{l_1, l_2} and it is called a factor of C with respect to the literals l_1 and l_2.Notice that the main difference between first order resolution and factorization is that the first one concerns two clauses while the second one concerns only one.

The concept of *proof* for first order resolution is analogous to propositional resolution but the clauses occurring are first order clauses, the resolvent is first order one and the definition must be supplemented by the condition that C_i in the proof can also be a factor of C_j for some $j < i$.

The inference rules *first order resolution* and *factorization* introduced above are called together as *unrestricted binary resolution* (in short *binary resolution*).

Now we state the completeness theorem for first order resolution:

2.31 THEOREM (completeness of first order resolution) *A set S of first order clauses with pairwise disjoint sets of individuum variables is unsatisfiable if and only if the empty clause $\square$ can be derived by binary resolution from S.*

2.3.5 Automatic theorem proving

Traditionally calculus has been considered a tool to test whether an argument is a correct proof. However, with the development of computing there has emerged the demand to create programs that generate proofs, that is, automatic theorem provers. The automatic theorem provers have been intended to be used in problem solving, program verification as well as to prove mathematical theorems.

We emphasize that, in general, a proof system does not provide a decision procedure, that is, the provability relation is not a decidable relation.A *proof system is not an algorithm, but it may be extended to an algorithm*. It only provides a *possibility* for a proof. Clearly, simple procedures can be written for the inference rules - they can be considered to be rewriting rules too[4]. However, to organize these procedures- that is the inference rules - into a process that proves theorems, we have to step outside logic. The automatic theorem provers work in the same way: the consequences of the input set of sentences are generated in some order, and the program tests whether the theorem to be proved is among them. R. Kowalsky ingenuity created a conceptual framework for this situation introducing the notion of the *proof procedure*. A proof procedure based on a proof system consists of a set of sentences, the set of inference rules and a *search strategy*. *The triple $\langle F_L\ , \vdash_C, Str\rangle$ of syntax, proof system and search strategy (Str) is called* a *proof procedure*.

Clearly, the sentences[5] and the inference rules that generate the consequences are provided by the logic in question; these two constituents generate the consequences of the input sentences, that is, the *search space*. The search strategy is a (maybe partial) ordering over the search space: it tells that in the actual state of the search space which rule has to be applied to which sentences (generated elements of the search space) with the purpose of generating new elements. But a search strategy *does not specify* the process of search in every step. As a second step towards automatic theorem proving, a theorem prover program "fills the gaps", turns the proof searching procedure into an algorithm. *A* theorem prover *can be conceived of an implementation of a proof procedure.*

Resolution calculus is considered a base of automatic theorem proving, moreover it was developed for that purpose - even now some consider resolution as the calculus for theorem provers. The reason is simple: the state of the art of

[4] The effect of computing is that the notion of "rewriting" has got into the logical terminology, see "relo".

[5] More precisely the syntax of the sentences.

computing technology of the sixties it was an important requirement to have easily representable structure; moreover, having only two very simple rules helped the construction of theorem provers. However, soon it has been realized that the search space can grow beyond control very fast - this phenomenon is called "*combinatorial explosion*". Several attempts have been proposed to overcome it. During the past decades, researchers have worked out several versions (the so called refinements) of resolution. They constrain the growth of the search space by constraining the condition of the execution of resolution rule. Some of the variants of resolution organize the clauses into some structure like connection graphs, or into tables [26]. These variants are very effective for ground clauses, however, they are usually lifted by the Davis–Putnam method with metavariables - which spoils effectiveness. Similarly some project concentrated on the search strategy: the general results in the area of search have been transferred into that of proof procedures. However, no adequate cost-function has been found. Several experiments have proven that there does not exist "best" resolution proof procedure, moreover a proof procedure that provides the best efficiency in the case of a task, may fail to generate the proof for another one.

For several decades resolution is not the proof system generally applied in theorem proving - Natural Deduction and related proof systems are also applied for theorem prover systems.

The most fruitful trend gives up automatic theorem proving and develops *proof assistants*. A proof assistant helps the user to describe a plan of the proof; it executes the prescribed steps and completes the proof automatically generated segments. The higher order proof assistant Isabelle is maybe the best example, for further information visit e.g. `http://isabelle.in.tum.de/`.

References to Section 2.3 are, for example: [51], [23], [49], [34], [163], [74], [35].

2.4 Topics from first-order model theory

Model theory is a quite modern area of mathematics, on the border of logic and universal algebra.

Historically, in mathematics fixed structures were investigated (e.g. the 3 dimensional space in geometry or the natural numbers in arithmetics, etc.). But, on the one hand, it turned out that, in general, axioms do not characterize completely a structure and; on the other hand, generalizations of classical mathematical areas were developed (spaces with metric, fields, topology, etc.). A new research area started: given an axiom system what kind of structures satisfy the axioms? Mathematicians started thinking about several models simultaneously. This was the beginning of model theory, too.

In what follows below, *model* will understood as a first-order model (first-order structure). Some problems from model theory: What is the relationship between different models of a given axiom system? How to generalize familiar algebraic concepts (subalgebra, homomorphism, generating system, etc.) to models? How to construct models in terms of other models (products, embeddings, etc.) and when does the new model preserve the properties of the old ones? Starting from a class of models of a given type, is it possible to axiomatize the class? What is the relationship of the syntactical properties of an axiom

system and the structural properties of the models satisfying this axiom system? How to characterize proof theoretical concepts (e.g. consistency, completeness, independence) in terms of models?

Today model theory is a very broad research area. In this section we outline only some selected areas and ideas which seem to be interesting for the purposes of this book.

First we treat the concept of *non-standard model* and the problem of *axiomatizability* of a given model. Then, we deal with theorems on *reducing satisfiability* of a given formula set to another formula set. Finally, a relatively new area of mathematics is considered: *non-standard analysis*.

2.4.1 Characterizing structures, non-standard models

One of the basic problems of logic is how to characterize a given structure, e.g., the geometry of the three-dimensional space or the structure of the real or natural numbers. For example, as is known, non-Euclidean geometries satisfy the "the minimal axiom system" of Euclidean geometry, so they can be considered as a "non-standard" model of the "Euclidean geometry".

As a generic example, first we deal with the structure of the natural numbers, and then we deal with arithmetics.

The language of arithmetics contains the function symbols 0, s (successor), $+$, $\cdot$ with arity 0, 1, 2, 2 respectively. The structure of natural numbers is: $\mathcal{N} = \langle N, 0^{\mathcal{N}}, s^{\mathcal{N}}, +^{\mathcal{N}}, \cdot^{\mathcal{N}} \rangle$, where N is the set of natural numbers, $0^{\mathcal{N}}$ is number zero, $s^{\mathcal{N}}$ is the successor operation, $+^{\mathcal{N}}, \cdot^{\mathcal{N}}$ are the familiar operations on N defined as usual. $Th\mathcal{N}$ is called the theory of *number theory* and $\mathcal{N}$ is called *standard model*.

Axioms of arithmetics ($Th\mathcal{N}$):

$\forall x(\neg s(x) = 0)$

$\forall x \forall y(s(x) = s(y) \to x = y)$

$\forall x(x + 0 = x)$

$\forall x \forall y(x + s(y) = s(x + y))$

$\forall x(x \cdot 0 = 0)$

$\forall x \forall y(x \cdot s(y) = x \cdot y + x)$.

The first two axioms ensure that any model of arithmetics must have an infinite universe. Partial recursive functions can be defined in the language, and computations with natural numbers can be performed using the axioms.

However, it is quite easy to construct structures satisfying these axioms in such a way that the basic properties of natural numbers (like commutativity) do not hold. Therefore, further axioms are needed.

For this purpose the *formula scheme for induction* is introduced:

$$\Phi(0) \wedge \forall x(\Phi(x) \to \Phi(s(x))) \to \forall x \Phi(x).$$

In this scheme Φ is a formula variable standing for first-order formulas with one free variable (so the schema represents infinitely many sentences, see Section 2.3).

The axiom system above together with the induction scheme is called *Peano's axiom system*, denoted by Pe.

A basic group of problems connected with $Th\mathcal{N}$ (as well as for a theory $Th\mathcal{A}$ of any structure $\mathcal{A}$) is the following one:

1. Is $Th\mathcal{N}$ axiomatized by Peano's axioms? If not, is it possible to complete this axiom system? If so, is it decidable?
2. To what extent is the structure $\mathcal{N}$ characterized by $Th\mathcal{N}$?

The following theorem which draws upon Gödel's and Church's famous results, answers the first group of problems:

2.32 THEOREM *$Th\mathcal{N}$ is not axiomatizable (not recursive enumerable) and not decidable.*

A *corollary* of this theorem is Gödel's incompleteness theorem: *$Ded\ Pe$ is incomplete*.

Otherwise $Ded\ Pe = Th\mathcal{N}$ would be recursive enumerable. Similarly, we get a negative answer for the other questions in 1.

Assume that the cardinality of the language is countable. For the second set of problems there is a general answer:

2.33 THEOREM (Löwenheim–Skolem–Tarski) *If a theory has an infinite model, it also has a model with a universe of cardinality λ for every infinite cardinal λ.*

Therefore, for any cardinality, $Th\mathcal{N}$ has a model of cardinality λ. For example, the axiom system Zermelo–Fraenkel's system of axioms (of set theory) has a countable model, too. In the light of the theorem, the second question can be sharpened as follows:

Does $Th\mathcal{N}$ have a model with a countable universe which is not isomorphic to $\mathcal{N}$?

The answer is affirmative. Models of $Th\mathcal{N}$ not isomorphic to the standard model are called the *non-standard models* of number theory. Elements of the universe of a non-standard model different from the natural numbers are called *non-standard numbers*. These definitions can be generalized:

If $\mathcal{A}$ is a model, then a model $\mathcal{B}$ of $Th\mathcal{A}$ which is not isomorphic to $\mathcal{A}$ is called a *non-standard model* of $Th\mathcal{A}$.

Let us summarize the answers for the problems 1 and 2 above: *$Th\mathcal{N}$ cannot be described* by the axiomatic method (contrary to the intuition of Leibniz,

Frege, Hilbert, and others). This famous result is referred to as a *limit* of proof theory. Another important result is this: even the complete theory $Th\mathcal{N}$ does not determine the structure $\mathcal{N}$ itself.

These results can be generalized from the number theory for theories being "strong enough". This latter means that recursive relations "are representable" in the theory. Most of the important theories used in mathematics have this property (real numbers, sets, etc.), therefore, these theories have non-standard models and these models are intensively investigated. Similarly, theories of abstract data types have non-standard models, as is shown below.

The following question is: *Is it possible to characterize the structure $\mathcal{N}$ in a unique way by choosing a suitable logic other than first-order logic?*

Second order logic is suitable for this purpose. Let us replace the following axiom for induction scheme in Peano's axioms:

$$\forall Z(Z(0) \wedge \forall x(Z(x) \to Z(s(x))) \to \forall x Z(x))$$

where Z is a relation variable. It can be seen that this new system of axioms characterizes $\mathcal{N}$ in a unique way in second-order logic.

But notice the difference between the second-order axiom and the Peano induction scheme. The first one states something about *all* relations defined on N, while the second one only about relations that can be *defined in the first order language* of number theory.

Now we encounter an example from data base theory. We deal with *abstract data types*. We meet a similar situation here as in the case of characterizing arithmetics.

Let us take the example of *stacks*, precisely stacks of numbers. A two-sorted language $\mathcal{L}_{stack}$ is considered with sorts *number* and *stack*. The language has the following non-logical symbols:

λ	$\langle$;stack$\rangle$	constant for the empty stack
push	$\langle$stack,number;stack$\rangle$	constructor function
pop	$\langle$stack,number,stack;$\rangle$	selector

Notice that the selector is given as a relational symbol for two reasons. The first is that the operator has two values, the second is that the selector is not defined for the empty stack.

The constant λ and the constructor function **push** generate the stacks that can occur in applications, these can be called *standard stacks*. The standard model **S** is a model such that S_{number} and S_{stack} consist of natural numbers and standard stacks, respectively. Functions and relations are defined in the standard way.

Let us see an axiom system for stacks, where x_i is a number variable and s_i is a stack variable.

The definition of the empty stack: $\forall x_1 \forall s_1 \neg \mathbf{pop}(\lambda, x_1, s_1)$.

The selector is a partial function:

$\forall s_1 \forall x_1 \forall x_2 \forall s_2 \forall s_3(\mathbf{pop}(s_1, x_1, s_2) \wedge \mathbf{pop}(s_1, x_2, s_3) \to x_1 = x_2 \wedge s_2 = s_3)$

The definition of the selector: $\forall x_1 \forall s_1 \mathbf{pop}(\mathbf{push}(s_1, x_1), x_1, s_1)$.

Equality of stacks:

$\forall s_1 \forall s_2(s_1 = s_2 \leftrightarrow \exists x_1 \exists s_3(\mathbf{pop}(s_1, x_1, s_3) \wedge \mathbf{pop}(s_2, x_1, s_3)))$

Induction scheme for stacks:

$$\Phi(\lambda) \wedge \forall s_1 \forall x_1 (\Phi(s_1) \to \Phi(\mathbf{push}(s_1, x_1))) \to \forall s_1 \Phi(s_1).$$

Notice that the induction scheme is very similar to the one of arithmetics. In fact, for almost every abstract data type an induction scheme of the same structure can be given.

Clearly there are non-standard models of the axioms, and in these structures the non-standard stacks are infinitely deep. Notice that in first-order logic we cannot express that "finitely many applications of pop reach the empty set", because the logical content of the phrase "finitely many" cannot be expressed.

2.4.2 Reduction of satisfiability of formula sets

We stated in Section 2.1.3 that first-order logic is compact. We formulate now a version of the Compactness theorem. Let Σ be a set of first-order formulas.

2.34 THEOREM (compactness of first-order logic) *Σ is satisfiable if and only if every finite subset of Σ is satisfiable.*

Compactness has many important applications. Such an application can be found in Section 2.4.3 (non-standard analysis). For another example we mention the following proposition:

The property "finite" can not be formulated in terms of a first-order formula (being finite is not a "first-order property").

To see this, let us assume that the meaning of a first-order formula α is "a set is finite". Consider now, for any n the following formula:

$$\varphi_n =$$
$$= \exists v_1 \exists v_2 \ldots \exists v_{n-1} (v_1 \neq v_2 \wedge v_1 \neq v_3 \wedge \ldots v_1 \neq v_n \wedge v_2 \neq v_3 \wedge \ldots \wedge v_{n-1} \neq v_n)$$

("the size of a set is larger or equal than n") and let us choose the set

$$\{\varphi_n \mid n \in \omega\} \cup \{\alpha\}\,.$$

By applying the compactness theorem, we can also prove for the set Σ in the compactness theorem above. Now, on the one hand, every finite subset of Σ is satisfied (there is a finite set with size $\geq n$). On the other hand, by compactness, Σ has a model $\mathcal{A}$ which is infinite, by construction. This obviously contradicts the finiteness property expressed by α.

Similarly, applying the compactness theorem we can prove the following propositions:

If a formula set Σ has a model with any finite cardinality, then Σ has an infinite model, too.

"Being a graph on the plane" is not a first-order property.

Yet another important type of reduction theorems are the Herbrand-type theorems. The point is here to reduce the unsatisfiability of a first-order formula to a set containing propositional formulas.

Assume that the formula in question is of the form $\forall x_1 \forall x_2 \ldots \forall x_m \alpha$, where the formula α is quantifier free. As is known, every first-order formula can be

transformed into such a form ("Skolem normal form"). Assume that there is a constant symbol in the language $\mathcal{L}$ and consider the closed formulas in $\mathcal{L}$ obtained from the formula α, from functions and constants of $\mathcal{L}$ step by step, i.e. consider the "ground instances" of α (see Section 2.3.3). Let $\{S_r \mid r \in Q\}$ denote the set of the ground instances of α. The ground instances are quantifier and variable free formulas, therefore, they obviously can be considered also as propositional formulas.

2.35 THEOREM (Herbrand) *$\forall x_1 \forall x_2 \ldots \forall x_m \alpha$ is unsatisfiable if and only if there is a finite unsatisfiable subset of the formula set $\{S_r \mid r \in Q\}$ in propositional logic.*

This theorem allows us to reduce unsatisfiability in first-order logic to propositional logic. This kind of reduction is important because handling formulas in propositional logic is easier than in first-order logic. The main application of Herbrand theorem can be found in resolution theory, i.e. in automatic theorem proving (Davis–Putnam method).

Both compactness theorem and Herbrand's theorem include a finitization proposition. Finitization has central importance in mathematics, but, in general, it is not a decision procedure.

2.4.3 On non-standard analysis

Today, non-standard analysis is a separated area within mathematics. Many theories exist for the topic. The extended logical theory of non-standard analysis uses *type theory*, in particular, Henkin's semantics. The theory is based on a *general compactness theorem* or, equivalently on *generalized ultraproducts*.

There is also an *axiomatic introduction* to non-standard analysis, making reference to logic unnecessary. This is used for example in non-standard probability theory or in non-standard measure theory (mathematical logic is present in an abstract level in these areas, see Boolean algebras).

In this section we do not go into the details of the general theory. We are going to discuss an interesting extension of the usual concept of real numbers, called *non-standard real numbers*. The non-standard models of real numbers are an important instance for the general concept of *non-standard models*. We show some basic ideas of non-standard analysis, using only first order logic. This is meant to be an elementary and didactic introduction, and in this respect we follow the textbook [51]. Some words on the history of the theory:

In the 17th century, Newton and Leibniz developed the differential and integral calculus using infinitely small, but non-zero quantities (so-called infinitesimals). In absolute value, these quantities are less than any usual positive real number (they may occur in denominators). Mathematicians heavily criticized Newton and Leibniz's Calculus because, among others, the concept of infinitesimals was not defined in a rigorous way from the viewpoint of mathematics. The exact foundations of the calculus were developed only in the 19th century (due to Bolzano, Weierstrass and others) introducing the so-called $\epsilon - \delta$ technique and eliminating the concept of infinitesimals from the theory. This theory was exact but it did not seem to be an ideal model for the concept of limit from an intuitive viewpoint.

In 1961, a logician and algebraist, Abraham Robinson lay an exact foundation for *infinitesimals*, using logic. Robinson applied his method to many other areas of mathematics (to algebra, set theory, topology, etc.), too. His *general theory* is also known as *non-standard analysis*.

The history of non-standard analysis is remarkable from the viewpoint of logic. The application of logic and the clarity of the logical concepts allowed mathematicians to develop the exact theory of infinitesimals.

Now we sketch some concepts of this theory:

By *real* here we mean a usual (standard) real number or else an attribute will be attached to the noun to warn the reader. Let $\mathcal{L}$ be a fixed language of real numbers such that $\mathcal{L}$ contains the most important real functions and relations and for every real i $\mathcal{L}$ contains a constant symbol c_i. So $\mathcal{L}$ is an extension of the usual language of reals and it should be considered to be as large as possible. Let ν denote the type of $\mathcal{L}$. Denote by $\mathcal{Q}$ the standard model of the reals of type ν with universe R, where R is the set of the usual reals. $Th\mathcal{Q}$ is called the standard theory of the real numbers.

2.36 Definition
Let $\mathcal{A}$ be an arbitrary model of type ν, with universe A. An element $r \in A$ is said to be infinitesimal *if $0 < |r| < c_i^{\mathcal{A}}$ for every positive real i, where $c_i^{\mathcal{A}}$ and $||$ are the interpretations of c_i and the magnitude in $\mathcal{A}$.*

An element $r \in A$ is said to be infinite *if $\left|c_i^{\mathcal{A}}\right| < r$ for every real i. An element is* finite *if it is not infinite.*

Let $\mathcal{A}$ and $\mathcal{B}$ be models of type ν. $\mathcal{B}$ is an elementary extension of $\mathcal{A}$ *if $\mathcal{A}$ is a submodel of $\mathcal{B}$ and for every formula $\alpha(x_1 \dots x_n)$ and elements $a_1 \dots a_n \in A$:*

$\mathcal{A} \vDash \alpha(a_1 \dots a_n)$ if and only if $\mathcal{B} \vDash \alpha(a_1 \dots a_n)$.

*In particular, $\mathcal{A}$ and $\mathcal{B}$ are "*elementary equivalent*" if $\mathcal{A} \vDash \alpha$ if and only if $\mathcal{B} \vDash \alpha$ for every* sentence *α.*

Elementary extension means that $\mathcal{B}$ inherits all the *first-order properties* of $\mathcal{A}$. Elementary extension is an instance of the important "transfer principle" in mathematics.

2.37 Theorem (Existence of infinitesimals) *There is a model $\mathcal{B}$ such that $\mathcal{B}$ is an elementary extension of the standard model $\mathcal{Q}$ of reals. In this case, there are infinitesimals in $\mathcal{B}$.*

Some remarks on the theorem:

- $\mathcal{B}$ is a *non-standard* model of $Th\mathcal{Q}$ because $\mathcal{B}$ cannot be isomorphic to $\mathcal{Q}$ since it contains infinitesimals. We introduce the following notational convention: if f is a concrete function or relation on the standard model $\mathcal{Q}$, then let ${}^*f$ denote the corresponding function or relation on $\mathcal{B}$, that is, the extension of f to $\mathcal{B}$. So f and ${}^*f$ are interpretations of the same function or relation symbol in the language $\mathcal{L}$.

- The *language* $\mathcal{L}$ plays an important role in dealing with non-standard models because it is a "bridge" between the standard model $\mathcal{Q}$ and non-standard model $\mathcal{B}$. Because of the property "elementary extension", the first-order properties of $\mathcal{Q}$ are inherited by $\mathcal{B}$. But, for example, the property "every bounded set has a least upper bound" is *not* a first-order property of $\mathcal{Q}$ and therefore, the satisfaction of this property on $\mathcal{B}$ is *not necessary*. In fact, this proposition is *false* on $\mathcal{B}$, since the set I of infinite numbers is bounded in $\mathcal{B}$ *by* any infinite number, but there is no least infinite number.

- The non-standard model is *not* a unique extension of standard reals, i.e. there is *no* "standard" non-standard model (while the original standard model is unique, by definition).

What is about the proof of the existence theorem? *One possible proof* of the theorem is an application of the *compactness theorem*, another one is an application of the concept of the algebraic construction "ultraproduct" (as is known, these two techniques are often equivalent). The first possible proof ensures only the existence of a non-standard model, the second one is stronger because it also yields a *construction* for this model. We sketch the main ideas of these proofs.

As regards the proof using the compactness theorem, consider the following set Σ of formulas:

$$\Sigma = Th\mathcal{Q} \cup \{0 < x \wedge x < c_i \mid i \in R\}$$

that is $Th\mathcal{Q}$ (the set of the truth propositions on R which can be formulated in terms of the language $\mathcal{L}$) is extended with the set of formulas $\{0 < x \wedge x < c_i \mid i \in R\}$ defining the infinitesimal x.

Both $Th\mathcal{Q}$ and any finite subset of $\{0 < x \wedge x < c_i \mid i \in R\}$ are obviously satisfiable on $\mathcal{Q}$. So the compactness theorem applies to Σ, and as a consequence, *$Th\mathcal{Q}$ has a model with infinitesimals* (and this model is obviously an elementary extension of $\mathcal{Q}$).

As regards the *other proof* of existence theorem we give an intuitive description, which does not require us to discuss the exact definition of ultraproducts.

The procedure of constructing non-standard reals from standard ones is similar to that of constructing reals from rational numbers. Recall that the reals are certain sequences of rationals, e.g. decimals (more exactly they are equivalence classes of certain sequences of rationals). The case for non-standard numbers is similar: the new real numbers (the non-standard numbers) are *certain sequences of standard reals* (more exactly they are equivalence classes of certain sequences of reals).

A zero-sequence of reals (a real sequence with limit zero) *represents a number infinitesimal.* A sequence of real numbers with "limit" *infinite* represents a non-standard *infinite* number. A constant sequence, i.e. a real sequence consisting of a given standard real i represents the *standard* number i.

Now we check the definition of *positive infinitesimal* (positive and less than any positive standard real) in terms of sequences of reals:

A zero-sequence of positive reals (that is a positive infinitesimal) must be "less" than any constant real sequence (a standard number). Roughly speaking, the meaning of "less" here is the following: "the ith member of the first sequence

is less than that of the second sequence, with the exception of *finitely many i*". This is satisfied by any zero- and constant sequence, as can be seen applying the concept of *convergence* of a zero-sequence.

A standard real function f can be considered as a function defined on the new numbers, i.e. on the members of a sequence of reals. This extension can be considered as ${}^*f$ defined on the non-standard numbers. Notice that it is an instance of the "transfer principle", too.

The following theorem allows us to visualize the world of the new numbers in a non-standard model $\mathcal{B}$.

2.38 Theorem (decomposition of non-standard numbers) *In a non-standard model $\mathcal{B}$ for every finite number r there is a standard real s and an infinitesimal p such that $r = s + p$. This decomposition is unique.*

Here s is called the *standard component of* r. This theorem shows that for every finite number there is a standard number which is "infinitely close" to r. "Infinitely close" means that their difference is infinitesimal, this relation being denoted by $\sim$.

Finally, as an illustration for the application in the Calculus, here is the definition of continuity and differentiability of a standard real function at a standard point a:

2.39 Definition
f is continuous *at a if and only if $x \sim a$ implies ${}^*f(x) \sim f(a)$ for any non-standard x (if x is infinitely close to a, then ${}^*f(a)$ is infinitely close to $f(a)$).*

f is differentiable *at a if and only if the standard part of the quotient $\frac{{}^*f(a+h)-f(a)}{h}$ is the same constant for every infinitesimal h (this constant is called the derivative at a).*

The other concepts and theorems of Calculus also can be formulated in a non-standard way, e.g. limit, integral.

References to Section 2.4 are, for example: [51], [23], [15], [117], [106].

Chapter 3

NON-CLASSICAL LOGICS

Considering the general definition of logic we can see that, in general, varying the set $\mathcal{F}$ of formulas or varying the meaning function m or varying other parameters of a logic we obtain new logics.

Logics which are not classical are called *non-classical* ones. We treat in this section such non-classical logics that seem to be interesting from the viewpoint of applications.

Setting out from classical logics and varying a given parameter mentioned above could be a starting point for classifying non-classical logics. Modal-, temporal-, arrow logics are obtained by extending the language, while we obtain many valued- and probability logic by extending the range of the meaning function. Intuitionistic logic is obtained among others by restricting the provability relation.

3.1 Modal and multi-modal logics

Modal propositional logic is a generalization of classical propositional logic. New unary logical constants will be introduced, which *modify the meaning* of propositions. The meaning of the original propositional formula can be modified applying the so-called modal operators $\Box$ and $\Diamond$ to the formula. For example, "P is true" can be modified as "it is necessary that P is true" or, "it is known that P is true" or, "it is believed that P is true", and all these modified meanings can be expressed by the formula $\Box P$. Another modification of "P is true" is "it is possible that P is true", which can be expressed by the formula $\Diamond P$.

Multi-modal propositional logic is a class of modal logics. Here there are many copies of the modal operator pair $\Box$ and $\Diamond$. These copies are indexed by elements of an index set I, they are $\Box_i$ and $\Diamond_i$ $(i \in I)$. Examples for multi-modal logics: temporal-, stratified-, dynamic, etc. logics.

Modal logic is one of the most important logics as regards possible applications. Modal logic plays an important role, among others, in *Artificial Intelligence*. Another application of modal logic is the investigation of executions of

a given computer program: $\Box\alpha$ can be interpreted so that "every terminating execution brings about α true" and $\Diamond\alpha$ can be interpreted so that "there is some execution which brings about α true". These readings of $\Box$ and $\Diamond$ are connected with *dynamic* logic.

Multi-modal logic is applied in Artificial Intelligence, too, for example, when we speak of several agent's beliefs or knowledges. In this case $\Box_i\alpha$ means that the ith agent believes or knows α. Important multimodal logics are: *dynamic* and *temporal* logics.

We sketch now the classical propositional modal logic L.

The *alphabet* of the language $\mathcal{L}$ of L is that of propositional logic supplemented by two new logical constants, which are the following two unary symbols: $\Box$ (box) and $\Diamond$ (diamond). To get the definition of a formula, that of a *propositional formula* is supplemented by the following: if α is a formula then, $\Box\alpha$ and $\Diamond\alpha$ are also formulas.

3.1 Definition *The* model *(or* frame*)* $\mathcal{C}$ *for* L *is defined as a triple*

$$\mathcal{C} = \langle W,\ S,\ \{C(p) \mid p \in W\}\rangle \tag{3.1}$$

where W *is any set (the set of the worlds),* S *is a binary relation on* W *(the* accessibility relation*), and* $C(p)$ *is a classical* interpretation *of the propositional symbols (a mapping from the set of propositional symbols to the set of truth values,* truth *and* false*).*

We are going to define the *meaning function* m in terms of the validity relation $\vDash$ (rather than defining validity in terms of meaning function).

We define $p \Vdash_{\mathcal{C}} \alpha$ (p *forces* α in $\mathcal{C}$), we will omit the subscript C from now on:

3.2 Definition

(i) *If* P *is a propositional symbol, then* $p \Vdash P$ *if* P *is true in* $C(p)$

(ii) *if* $p \Vdash \alpha$ *and* $p \Vdash \beta$ *are defined, then* $\neg\alpha$, $\wedge\alpha\beta$, $\vee\alpha\beta$, $\to\alpha\beta$, $\leftrightarrow\alpha\beta$ *is true* if and only if α *is false,* α and β *are true,* α or β *is true,* α *is false or* β *is true,* α *and* β *are* simultaneously *true or false, respectively,*

(iii) *if* $q \Vdash \alpha$ *is defined for all* $q \in W$*, then* $p \Vdash \Box\alpha$ $(p \in W)$ if and only if for all $q \in W$ *such that* pSq*,* $q \Vdash \alpha$*,*

(iv) *if* $q \Vdash \alpha$ *is defined for all* $q \in W$*, then* $p \Vdash \Diamond\alpha$ $(p \in W)$ if and only if there is $q \in W$ *such that* pSq *and* $q \Vdash \alpha$*.*

Notice that points *(i)*-*(ii)* are the same as for the classical proposition case.

3.3 Definition *Having the relation* $p \Vdash_{\mathcal{C}} \alpha$*, the value of the meaning function* m *for the formula* α *and frame* $\mathcal{C}$ *is the set* $\{w \in W \mid w \Vdash_{\mathcal{C}} \alpha\}$*. We say that* α is forced in the frame $\mathcal{C}$ *(or* α *is true in* $\mathcal{C}$*,* $\mathcal{C} \vDash \alpha$*) if* $w \Vdash_{\mathcal{C}} \alpha$ for every $w \in W$*.*

It is easy to check that the connection between the modal operations $\Box$ and $\Diamond$ is that $\Box$ is equivalent to $\neg\Diamond\neg$.

The definitions above are due to Carnap and Kripke.

There are important *specifications* of modal logic according to the properties of the accessibility relation S (see Definition 3.1). For example, if S is reflexive, transitive and symmetric (i.e. it is an equivalence relation), we obtain the modal logic $S5$, and if the symmetricity is not required here, we obtain the logic $S4$.

In a frame of $S5$ every world is accessible from any other one, therefore, sometimes S will not be indicated. Then, the definition of $\Box$ is: $p \Vdash \Box\alpha$ $(p \in W)$ if *for all* $q \in W$, $q \Vdash \alpha$. Restricting $S5$ to classical propositional language we can associate a modal frame of $S5$ with the collection of models of classical propositional logic. So the semantics of classical propositional logic can be defined as a modal semantics.

Concerning the *proof theory* of modal logics there are strongly complete and sound inference systems for modal logics, among others for $S4$ and $S5$. Considering the Hilbert system for propositional logic, besides modus ponens an additional inference rule, the "necessitation rule" $\langle\langle\Phi\rangle, \Box\Phi\rangle$ is needed. Furthermore, a new axiom is also needed (the so-called K axiom: $\Box(\Phi \to \Psi) \to (\Box\Phi \to \Box\Psi)$). Many properties of S can be postulated by *axioms*, e.g. reflexivity can be postulated by the scheme $\Phi \to \Box\Diamond\Phi$, transitivity by the scheme $\Box\Phi \to \Box\Box\Phi$, etc.

We note that general modal logic has *no Deduction Theorem* with respect to $\to$, in general. For example, $\alpha \to \Box\alpha$ is not valid but $\alpha \vDash \Box\alpha$ holds. However some modal logics have Deduction theorem, e.g. in $S4$, $\Sigma \cup \{\alpha\} \vDash \beta$ if and only if $\Sigma \vDash \Box(\alpha \to \beta)$.

As regards the *intuitive meanings* of the modal operator $\Box$, there are several approaches.

The basic meaning of $\Box$ is: "it is necessary that" (for $\Box\alpha$: "it is necessary that α is true"). In Kripke's terminology this means in a frame that "it is true *in every* possible world w, accessible from the actual world". Similarly, the meaning of $\Diamond$ is "it is possible that" and this means in a frame that "it is true at least in one accessible possible world w". In these contexts S must be assumed to be reflexive.

Another possible meaning of $\Box$ is "it is known" (for $\Box\alpha$: "it is known, that α is true") or another one is: "it is believed". Here S must be assumed to be reflexive, transitive. With these meanings of modal operators we can associate modal logics which are "between" $S4$ and $S5$.

A further possible reading of $\Box\alpha$ is "α will be true at all future times". This meaning is connected with *temporal logic*, which we discuss later.

***

We outline *first-order modal logic* referring only to the differences in comparison with the propositional case:

To get the syntax of the first-order case, the alphabet of classical first-order logic is extended by the operators $\Box$ and $\Diamond$. The definition of formulas is the usual.

Definition 3.1 of a modal frame is modified so that $\mathcal{C}(p)$ means a usual first-order model. Let $C(p)$ denote the universe of $\mathcal{C}(p)$.

While the definition of a first-order modal model is straightforward, to define the forcing relation we have to solve certain problems. Several versions exist. The problems are: how to define the interpretations of *terms* and how to define the meaning of quantifications? These two problems are obviously closely related to each other.

If only one possible world is considered, the interpretation of a term is defined as in first-order logic. In the case of several worlds the problem is whether the interpretations of terms should coincide in the worlds accessible from a given world in question.

A term is called rigid *for a modal frame C if its interpretations coincide in the worlds in C.* There are two basic types of rigidity:

(i) every term is rigid

(ii) variables are rigid, but the other terms are not.

We can associate different meanings (definitions) of quantification with the above-mentioned different types of rigidity (including the case of non-rigidity). We do not discuss here the possible definitions of quantification.

It is hard to find a complete calculus for first-order modal logic. Completeness depends on the definition of quantifiers and the accessibility relation. For modal logics $S4$ and $S5$ completeness can be proven.

We note that rigidity and the condition $C(p) \subseteq C(q)$ are quite restrictive (e.g. for temporal logic, where objects often come to exist and perish). It is usual to introduce a special non-logical predicate with the intended meaning "alive". The intended meaning has to be described by axioms.

***

Multi-modal logics are generalizations of the simplest modal logic. We only list the differences between modal and multi-modal logics.

As we have mentioned, the *alphabet of the language* of this logic contains a set of pairs of modalities denoted by $\Box_i$ and $\Diamond_i$ $(i \in I)$.

In the recursive definition of *formulas* the following modification is needed: if α is a formula, then $\Box_i\alpha$ and $\Diamond_i\alpha$ are also formulas.

A *frame* here is the triple: $\mathcal{C} = \langle W, \{S_i \mid i \in I\}, \{C(p) \mid p \in W\}\rangle$, where $\{S_i \mid i \in I\}$ is the set of the accessibility relations, corresponding to the operators $\Box_i$ and $\Diamond_i$.

The definition of $p \Vdash_{\mathcal{C}} \alpha$ is modified so that instead of $\Box$, $\Diamond$ and S, the modalities $\Box_i$, $\Diamond_i$ and the relation S_i are used. The definition of the meaning function m is analogous with the definition given earlier.

3.2 Temporal logic

Temporal logic is a special multi-modal logic. Usage of temporal logic allows us to manipulate relations pertaining to time, without defining the concept *of time*.

In *temporal logic* there are two pairs of modal operator symbols, the *future operators* F and G and the *past operators* P and H, where F and P are operators of type "box", G and H are operators of type "diamond". The meanings of the future operators F and G applied to formula R are: "R will be *always* true" and "R will be true at *some* time", respectively. The meanings of the past operators P and H: "R has always been true" and "R has been true". These are the usual temporal quantifiers. In different versions of temporal relations other quantifiers are also used expressing certain temporal relations, such relation is e.g. "until".

Temporal logic is *applied* mainly in the specification and verification of dynamic systems such as computer hardware and software. It plays an important role in the specification and verification of concurrent programs and hardware components, too. Some kinds of temporal logic (e.g. McDermott's logic, Allen's logic) enable one to prove results about facts, events, plans or history. These applications have importance, among others, in artificial intelligence.

Now we sketch some related concepts of temporal logic.

3.4 Definition *A* temporal frame *is the sequence*

$$\mathcal{C} = \langle T,\ S_F,\ S_P,\ \{C(t) \mid t \in T\}\rangle$$

*where T is a set of points of time, S_F and S_P are the accessibility relations (*temporal precedences*) corresponding to the future operator F and the past operator P respectively, furthermore, $S_P = S_F^{\smile}$ is assumed (where $^{\smile}$ is the converse operator, so sS_Pt is true if and only if tS_Fs is true).*

Since $S_P = S_F^{\smile}$, a temporal frame is usually defined in a simplified form

$$\mathcal{C} = \langle T,\ R,\ \{C(t) \mid t \in T\}\rangle$$

where $R = S_F$ (therefore, $S_P = R^{\smile}$).

Using the general Definition 3.2 (*(iii)* and *(iv)*), we get

$$s \Vdash F\alpha\ (s \in T) \text{ if } t \vDash \alpha \text{ for all } t \in T \text{ such that } sRt \tag{3.2}$$

$$s \Vdash G\alpha\ (s \in T) \text{ if } t \vDash \alpha \text{ for some } t \in T \text{ such that } sRt \tag{3.3}$$

Similar definitions apply for P and H. Obviously the meanings of 3.2 and 3.3, respectively: s forces $F\alpha$ in $\mathcal{C}$ if α is true for *every* future times, and s forces $G\alpha$ in $\mathcal{C}$ if α is true for *some* future time.

If constraints are assumed for the temporal precedence relation we obtain different versions of temporal logic. The logic *without* these restrictions is called *minimal* temporal logic.

The classical picture of time is that time is a *linear* continuum (for example, in physics). We need two restrictions on R to express linearity. These are *transitivity* and *trichotomy*, respectively:

$$\forall t \forall s \forall r((Rts \wedge Rsr) \to Rtr) \tag{3.4}$$

$$\forall t \forall s \forall r(Rts \vee t = s \vee Rst). \tag{3.5}$$

Time can also be modelled as "*branching time*", where we have to postulate the property of "*backwards linearity*" (or "*left linearity*").

$$\forall t \forall s \forall r((Rtr \wedge Rsr) \rightarrow Rts \vee t = s \vee Rst)$$

rather than trichotomy.

Here for any given time, several different futures are possible. For example, using "branching time" we can model the execution of a non-deterministic algorithm. "Forwards linearity"(or "right linearity") can be defined in a similar way.

Further possible properties of time are those of ordering: ordering with or without endpoints, dense or discrete ordering, etc.

We skip the details of the possible proof systems for temporal logics. Basic temporal logic (as a multimodal logic) *has a complete proof system*. Regarding the extensions of basic temporal logic, some of them are axiomatizable (e.g. the model of transitive time), while some of them are *not* (e.g. the model of discrete time).

First-order temporal logic can also be defined also as a special first-order modal logic.

3.3 Intuitionistic logic

Intuitionistic logic is the most important by-product of intuitionistic mathematics. Intuitionists hoped to eliminate all non-constructivist tools from mathematics. Sometimes we do not realize that intuitionistic logic is used. Such a logic is, for example, the "logic of Horn formulas", which is used in logic programming. To motivate the discussion that follows,we outline some principles of intuitionistic mathematics (and also that of constructivism).

Let us first consider the *constructive notion of proof*. This notion is interesting for us because it brings us closer to the theory of computation. Considering the clauses below one can see what it means to prove a non-atomic formula φ in the terms of the proofs of its components:

1. $\alpha \wedge \beta$ *is proven iff* α *is proven and* β *is proven,*
2. $\alpha \vee \beta$ *is proven iff* α *is proven or* β *is proven,*
3. $\alpha \rightarrow \beta$ *is proven iff every proof of* α *can be converted into a proof of* β*,*
4. $\exists x \alpha(x)$ *is proven iff for some element* a *of the domain of quantification* $\alpha(x/a)$ *is proven,*
5. $\forall x \alpha(x)$ *is proven iff for every element* a *of the domain of quantification* $\alpha(x/a)$ *is proven.*

These principles provide the simplest version of *constructionism*. From the point of view of computing, clause (4) is the most important one: *to prove the existence of an element satisfying a specification, such an element has to be selected*. The development of several logics is motivated by constructionism, and from the viewpoint of computer science the most important one is Martin Löf's type theory (see e.g. [156]).

Intuitionistic first-order logic meets the principles listed above. The *syntax* of intuitionistic logic is the same as that of first-order languages, however, a new symbol $\bot$ meaning "false" is *introduced* and, as a consequence, negation can be expressed in terms of implication. In particular, formulas of form $\neg\alpha$ can be rewritten into $\alpha \to \bot$.

Regarding *calculus*, a calculus for intuitionistic logic can be obtained from classical first-order calculus by omitting some axioms and rules. The following proposition holds: *Intuitionistic logic is "conservative" with respect to classical logic in the sense that every formula that is provable in intuitionistic logic can already be proven in classical logic.*

For example, the following sentences can be proven in intuitionistic calculus: $\alpha \to \neg\neg\alpha$, $\neg(\alpha \wedge \neg\alpha)$, $\neg(\alpha \vee \beta) \leftrightarrow (\neg\alpha \wedge \neg\beta)$, $\neg(\alpha \wedge \beta) \leftarrow (\neg\alpha \vee \neg\beta)$, $(\alpha \to \beta) \to (\neg\beta \to \neg\alpha)$, $\exists x\neg\alpha(x) \leftarrow \neg\forall x\alpha(x)$ etc.

However, several important tautologies of classical logic cannot be proven in intuitionistic logic, some of the most important ones are:

- $\neg\alpha \vee \alpha$ (excluded middle),
- $\neg(\alpha \wedge \beta) \to (\neg\alpha \vee \neg\beta)$ (a part of deMorgan's Laws),
- $(\neg\alpha \to \neg\beta) \to (\beta \to \alpha)$,
- $\neg\forall x\alpha(x) \to \exists x\neg\alpha(x)$.

It can be seen that instead of some equivalences only implications can be proven. The reason is that in an intuitionistic calculus, it is more difficult to prove negated statements than positive ones. Similarly it is more difficult to prove formulas with $\vee$ than in classical logic, because to prove $\alpha \vee \beta$ we have to prove α or β.

Maybe the most important difference between intuitionistic and classical logics is that the connectives and quantifiers do not depend on each other. Therefore, there are no conjunctive or prenex normal forms ((the latter is true because the sentence $\neg\forall x\alpha(x) \to \exists x\neg\alpha(x)$ is not provable in intuitionistic logic).

For intuitionistic logic different *semantics* have been worked out, the simplest and most intuitive one is the Kripke's semantics (see below) which is based on first order modal frames.

3.5 Definition (intuitionistic model) *A model $\mathcal{C}$ of intuitionistic logic is a model (frame) of modal logic:*
$\langle W, S, \{C(p) : p \in W\}\rangle$ satisfying the following conditions:

(i) *S is a partial ordering with a minimal element,*

(ii) *if p is accessible from q (qSp holds), then the universe of $C(q)$ is a subset of the universe of $C(p)$,*

(iii) *if p is accessible from q (qSp holds), then all the atomic formulas true in $C(q)$ must be true in $C(p)$, too.*

Clearly the third condition holds for every positive formula (in a positive formula connectives $\to$ and $\neg$ do not occur).

Let $\langle \mathrm{W}, \mathrm{S}, \{C(p) : p \in W\}\rangle$ be an intuitionistic model. The property "$p \in W$ forces a formula α in $\mathcal{C}$", that is $p \Vdash \alpha$ is defined by recursion as follows:

3.6 Definition (forcing relation)

(i) *if* α *is atomic, then* $p \Vdash \alpha$ *iff* $C(p) \vDash \alpha$,

(ii) $p \Vdash \alpha \wedge \beta$ *iff* $p \Vdash \alpha$ *and* $p \Vdash \beta$,
$p \Vdash \alpha \vee \beta$ *iff* $p \Vdash \alpha$ *or* $p \Vdash \beta$,

(iii) $p \Vdash \alpha \to \beta$ *iff for every* $q \in W$ *if* pSq *and* $q \Vdash \alpha$, *then* $q \Vdash \beta$,
$p \Vdash \neg\alpha$ *iff for every* $q \in W$ *if* pSq, *then* $q \Vdash \neg\alpha$,

(iv) $p \Vdash \forall x\alpha$ *iff for every* $q \in W$ *if* pSq, *then for every element* a *of the universe of* $C(q), q \Vdash \alpha(x/a)$,

(v) $p \Vdash \exists x\alpha$ *iff there exists an element* a *of the universe of* $C(p)$ *such that* $p \Vdash \alpha(x/a)$.

Concerning *(iii)* if $\neg\alpha$ is considered as a short form of $\alpha \to \bot$, then the fact that no $p \in W$ forces $\bot$ implies the following: $p \Vdash \neg\alpha$ if and only if for no $q \in W$ such that pSq and $q \Vdash \alpha$.

The *meaning function* can be defined as in the basic modal case. Then, the *truth on a model*: $W \vDash \alpha$ if and only if all $p \in W$ forces the sentence α.

Calculus can be introduced for first-order intuitionistic logic. The following completeness property is true:

Intuitionistic first-order calculus is strongly complete for the above Kripke semantics.

Notice that a first-order intuitionistic model is a *special* first-order modal frame. In Definition 3.6 the condition of forcing for formulas $\alpha \to \beta$ and $\forall x\alpha$ is the same as in the case of first-order modal logic. So the *semantics of intuitionistic logic can also be given by translating it into first-order modal logic*, where the class of frames is restricted to the frames corresponding to the models of intuitionistic logic. The transformation can be performed by writing the modal quantifier $\Box$ before every subformula of the form $\alpha \to \beta$, $\neg\alpha$ and $\forall x\alpha$. Then, an intutionistic sentence α is true on an intuitionistic model $\mathcal{C}$ if and only if α translated into a first-order modal logic is true on $\mathcal{C}$ as a frame in modal logic.

Some concluding remarks concerning intuitionistic logic:

Compactness is the consequence of completeness.

The *Deduction Theorem* applies to intuitionistic logic in the same form as to the classical first-order one: $\Sigma \cup \{\alpha\} \vDash \beta$ is equivalent to $\Sigma \vDash \alpha \to \beta$, because of the $\to$ introduction rule and completeness.

Having completeness it is easier to prove that a sentence is not provable in intuitionistic logic: it is enough to prove that it is not universally valid, that is to give a model that does not satisfy the formula.

3.4 Arrow logics

Manipulating relations is in the focus in many areas of applications, e.g. in the theory of Databases. Arrow logics is based on the fundamental operations with relations besides propositional connectives, to establish a kind of logic of relations.

We sketch two types of arrow logics: Relation logic and Logic of relation algebras.

3.4.1 Relation logic (*RA*)

The alphabet of the language is: $\wedge$, $\neg$, and a binary operation symbol $\circ$ (*composition*) as logical connectives and the infinite sequence $P_1, P_2 \ldots$ of propositional constants.

The definition of formulas is the same as that of propositional formulas with the additional condition: if α and β are formulas, then $\alpha \circ \beta$ is also a formula.

3.7 Definition *The* model *of relation logic is the same* $\langle W, v \rangle$ *as that of propositional logic mentioned at modal logic* $S5$ *with the restriction that the set* W *is of the form* $U \times U$ *for some set* U *(so the elements of* W *are ordered pairs* $\langle u_1, u_2 \rangle$*, where* $u_1, u_2 \in U$*).*

The definition of the operation $\circ$:

3.8 Definition $\langle u_1, u_2 \rangle \vDash \alpha \circ \beta$ *if* $\langle u_1, u \rangle \vDash \alpha$ *and* $\langle u, u_2 \rangle \vDash \beta$ *for some* $u \in U$*, where* $\langle u_1, u \rangle$, $\langle u, u_2 \rangle \in W$*.*

The definitions of the *meaning function* and *validity relation* are the same as in the propositional case.

It can be proven that *theories* of the models (or model classes) may be not decidable for this logic. The main reason is the hidden quantifier in the definition of the operation $\circ$. But relation logic itself is decidable.

3.4.2 Logic of relation algebras

This logic is an extension of RA.

The alphabet of RA is extended by two new logical connectives: by a unary operation symbol $^\smile$ (*converse*) and the identity relation symbol Id.

In the definition of *formulas* the following two additional stipulations are needed:

If α *is a formula, then* $\alpha^\smile$ *is also a formula,*
Id *is a formula.*

The concept of *model* coincides with that of a model of RA.

The definitions of the operations $^\smile$ and Id:

3.9 Definition $\langle u_1, u_2\rangle \vDash \alpha^\smile$ *iff* $\langle u_2, u_1\rangle \vDash \alpha$ *for some* $\langle u_2, u_1\rangle$
$\langle u_1, u_2\rangle \vDash Id$ *iff* $u_1 = u_2$.

Definitions of the *meaning function* m and the *validity relation* are the same as in RA and in propositional logic.

It can be proven that this logic is *not decidable*.

The so-called *relation set algebras* are "algebraizations" of the logic of relation algebras. These algebras are "algebraization" of this logic.

Let U be a given set.

3.10 Definition *The relation set algebra* $\mathcal{R}$ *with base* U *is the structure*

$$\mathcal{R} = \langle A,\ \cup,\ \cap,\ \sim_{E_U},\ E_U,\ \emptyset,\ \circ,\ \smile,\ Id\rangle$$

where $\cup$, $\cap$, *and* $\sim_{E_U}$ *denote the usual union, intersection and complementation,* E_U *is an equivalence relation on* U *(so* $E_U \subseteq U \times U$*),* $\circ$ *is the operation relation composition,* $\smile$ *is the operation converse,* Id *is the identity relation restricted to* E_U *and the universe* A *is assumed to be closed under all the operations included in* $\mathcal{R}$.

3.5 Many-valued logic

We survey two types of many-valued logics: Kleene's and Łukasiewicz's logic.

First we discuss *Kleene's logic*. For it a new truth value will be introduced denoted by "u". The meaning of "u" will be "*undefined*".

The motivation of Kleene's logic is in the theory of recursive functions, among others. Namely, a machine may not provide answers (true or false) for certain inputs because of going into an infinite loop or of its limited capacity, etc.. This situation can be expressed by an *undefined* truth value u.

The language of this logic contains the logical connectives $\neg$, $\wedge$, $\vee$ and $\rightarrow$. The set of formulas is that of propositional formulas formed by these connectives. Let $\mathcal{P}$ denote the set of atomic formulas.

3.11 Definition *The class* $\mathcal{M}$ *of models consists of functions* $\{g_M : \mathcal{P} \to \{t, f, u\}\}$, *where* t, f *and* u *denote the truth values (called true, false, and undefined or not known) and the (natural) ordering on these truth values is:* $f < u < t$.

The definition of the *meaning function* m is recursive and similar to the modal or propositional case. Assume that the model M (and g_M) is fixed. Then

- if α is an atomic formula, then $m(\alpha)$ is defined,

- if the values of $m(\alpha)$ and $m(\beta)$ are defined for the formulas α and β and these truth values are t or f, then the definition of m for $\neg$, $\wedge$, $\vee$ and $\rightarrow$ is the same as for propositional logic,
- if one of the values of $m(\alpha)$ and $m(\beta)$ is u, then the result is u except for the cases $f \wedge u = u \wedge f = f$, $t \vee u = u \vee t = t$ and $u \rightarrow t = f \rightarrow u = t$.

Notice that there are no tautologies in this logic, and, in particular, $\alpha \rightarrow \alpha$ is *not* a tautology because of $u \rightarrow u = u$.

The relation $\models$ is defined so that $M \models \alpha$ if and only if $m(\alpha) = t$, as usual.

Another many-valued logic is *Łukasiewicz's 3-valued logic*. The only *formal* difference between Łukasiewicz's logic and Kleene's logic relates to the connective conditional $\rightarrow$. Now let us denote the third truth value by i rather than by u.

Let the definition of $i \rightarrow i$ be t in Łukasiewicz's logic.

A consequence of this definition is that each instance of the scheme $\alpha \rightarrow \alpha$ is a tautology.

Apart from this definition, the *essential* difference between these two logics is in their *motivations*. The meaning of i could be the claim that a statement will be true in the *future* (contrary to the fatalist position that a statement about the future is *now* either true or false). In this case a statement is neither true nor false but it is *undetermined* (not only that its truth value is not known but it *does not have it*). This logic can be regarded as the "logic of definitions".

There are known generalizations of Łukasiewicz's logic to n truth values ($n \geq 3$) and to infinitely many values.

We note that the definition of semantics of *first-order* 3-valued logic is analogous with that of classical first-order logic, based on a 3-valued propositional logic rather than on classical propositional logic.

***

Now we present an *example* to illustrate how a many-valued logic can be used to study problems from real life.

3-valued first-order logic seems to be suitable to describe the activity of certain intelligent robots. We assume that the robot is moving between states, collecting facts about its environment. In general, it collects data, its knowledge is incomplete. We assume that it never revises or discards its belief.

We can describe the activity of the above robot modifying Kleene's 3-valued *first-order* logic L, by taking the partial ordering in which u is less than either f or t.

Let the models of L represent the states of the robot. Let us fix the partial ordering $u < f$, $u < t$, denoted by $\leq$ on the set $\{t, f, u\}$ of the truth values. Let $\mathcal{M}$ be the class of models of L. Assume that $\mathcal{A}$ and $\mathcal{A}'$ are models of L with the same universe A.

The model $\mathcal{A}'$ is said to be an *extension* of the model $\mathcal{A}$ if

$$R^{\mathcal{A}}(a_0, \dots a_{n-1}) \leq R^{\mathcal{A}'}(a_0, \dots a_{n-1}) \tag{3.6}$$

for every relation symbol R of the language and elements $a_0, \dots a_{n-1}$ in A, in notation: $\mathcal{A} \preceq \mathcal{A}'$.

The following *monotonicity proposition* states that property (3.6) remains true for all the sentences of the language. Let $\alpha^{\mathcal{A}}$ denote the truth value of the sentence α in the model $\mathcal{A}$.

Proposition: $\mathcal{A} \preceq \mathcal{A}'$ implies $\alpha^{\mathcal{A}} \leq \alpha^{\mathcal{A}'}$ for every sentence α.

Let h be an operation defined on the models of L. If $\mathcal{A} \preceq \mathcal{A}'$ implies $h(\mathcal{A}) \preceq h(\mathcal{A}')$, then h is said to be *monotonic*. The following *fixed point proposition* is true:

Proposition: For any model $\mathcal{A}$ of L such that $\mathcal{A} \preceq h(\mathcal{A})$ there exists the least model $\mathcal{A}'$ such that $\mathcal{A} \preceq \mathcal{A}'$ and $h(\mathcal{A}') = \mathcal{A}'$.

$\mathcal{A}'$ is said to be the *fixed point of* h. The function h represents the transition of the robot from one state to another. The monotonicity theorem reflects the requirement that the robot should not revise or discard its belief. The fixed point $\mathcal{A}'$ represents the complete information that can be obtained by the procedure represented by h (but some things may remain undecided).

3.6 Probability logics

3.6.1 Probability logic and probability measures

By modus ponens, α and $\alpha \to \beta$ together imply β. A familiar incorrect inference is to conclude α from $\alpha \to \beta$ and β. This inference is incorrect, but it occurs in "real life", especially in diagnostic problems. For example, if A means that "all the driving mechanisms of an aeroplane stop working during flying" and B means that "the aeroplane crashes", then $A \to B$ can be considered as true. therefore, if A is satisfied, then B is also true, by modus ponens. But conversely, if we know only the truth of B, then we must not conclude the truth of A. But the following question arises: Assumed the truth of B and $A \to B$ what is the *chance* (probability) of A being true?

Drawing a conclusion from $A \to B$ and B to A (somehow) is a special kind of inference, an inference "backwards" (retroduction, inductive reasoning). In general, inferences like this very often occur in real life and they are a subject of probability logic and probability theory, too. Obviously, there is a close connection between probability logic and probability theory, we return to this connection in the next section. From the viewpoint of logic, investigating these kind of inferences means to investigate *probabilities defined on logical formulas*. The problem above is connected with propositional formulas and propositional logic. We discuss here a more general case: a probability logic related to *first-order logic*.

Probability logic can be considered as a logic with infinitely many truth values, which are real numbers between 0 and 1. There is a wide scale of probability logics. Here we introduce a logic that is based on ordinary first-order logic and its models are first-order models equipped with a probability measure. The proof theory of probability logic is strongly limited, this is why we do not discuss it here.

Now we sketch some concepts:

The set of *formulas* is the same as in first-order logic. For the sake of simplicity we assume that the language has no function symbols but has constant symbols. Denote by $\mathcal{F}$ the set of formulas.

We are going to define the concept of a "probability model", that is, the concept of *model* for probability logic.

Let M be a given first-order model of type of $\mathcal{L}$ with universe U and let $[\alpha]$ denote the *truth set* of α. The truth sets on M form a so-called "cylindric set algebra" $\mathcal{B}$ *with* unit U^ω (this fact will be discussed in Section 4.2)

3.12 DEFINITION *A non-negative real function P defined on the sets $\{[\alpha] \mid \alpha \in \mathcal{F}\}$ is said to be a* probability *on the given first-order model M if*

(i) $0 \leq P[\alpha] \leq 1$ *for every* α,

(ii) $P[\alpha] = 1$, *if* $[\alpha] = U^\omega$,

(iii) $P[\alpha \vee \beta] = P[\alpha] + P[\beta]$, *if* $[\alpha] \cap [\beta] = \emptyset$.

It can be proven that the definition of P does not depend on the formula α representing the set $[\alpha]$. The probability on M can be considered as a *usual probability measure defined on the Boolean part of the cylindric set algebra $\mathcal{B}$ corresponding to M*. Intuitively, $P[\alpha]$ means the probability that a randomly selected evaluation satisfies α in M.

In the above definition there is no quantifier and only the Boolean part of $\mathcal{B}$ is used. We extend the definition of probability so that it should reflect the presence of quantifiers. As is known (see [23]) in algebras corresponding to first-order models, the following relation is true

$$[\exists x \alpha] = \sup_n \bigcup_{i \in n} [\alpha(x/x_i)] \tag{3.7}$$

where x_i is not free in α, $\bigcup$ means Boolean supremum, that is these algebras are *closed under the suprema corresponding to existential quantifiers*. A similar relation is valid for universal quantifiers, too.

We use (3.7) to define the concept of *quantifier probability* (for short, Q-probability) on a model M.

3.13 DEFINITION *A probability P defined on the model M is said to be a* quantifier probability *(Q-probability) if the following property is satisfied for every formula α and individuum variable x:*

$$P[\exists x \alpha] = \sup_n P\left[\bigvee_{i \in n} \alpha(x/x_i)\right] \tag{3.8}$$

where $x_1, x_2, \ldots$ is the sequence of the individuum variables being not free in α.

Condition (3.8) means a kind of continuity of the probability, i.e. a *partial continuity* (with respect to the sums in (3.7) and also to the products corresponding to universal formulas), reflecting the *presence of quantifiers* in the logic.

Quantifier probabilities *can be constructed* on M by restricting well-known product measures defined on U^ω. Assume that the universe U of the model M is countable.

3.14 THEOREM *Let μ be a σ-additive probability on U and consider the infinite power of μ on the power σ-algebra with unit U^ω. Then the restriction of this power of μ to the cylindric set algebra corresponding to M is a quantifier probability.*

3.15 DEFINITION *A model K for probability logic is a first-order model M equipped with a quantifier probability P defined on M. The value of the meaning function for a formula α and model K is the real number $P[\alpha]$ where the ordering is that of the reals.*

For the validity relation: $K \vDash \alpha$ *holds if and only if* $P[\alpha] = 1$ *in* K.

We can define probabilities (and quantifier probabilities) directly on the set $\mathcal{F}$ of formulas of the language, too. This is the base of a possible proof theory for probability logic (not detailed here).

For example, the definition of probability on $\mathcal{F}$ is:

3.16 DEFINITION *A non-negative real function P defined on the set $\mathcal{F}$ of formulas is said to be a probability on $\mathcal{F}$ if for every formulas α and β the following conditions are satisfied:*

(i) $0 \le P(\alpha) \le 1$,

(ii) $P(\alpha) = 1$, *if* $\vDash \alpha$,

(iii) $P(\alpha \vee \beta) = P(\alpha) + P(\beta)$, *if* $\vDash \neg(\alpha \wedge \beta)$.

Consequences of this definition are: $P(\alpha) = P(\beta)$ if $\vDash \alpha \leftrightarrow \beta$ and $P(\neg\alpha) = 1 - P(\alpha)$.

It is easy to see that a probability on $\mathcal{F}$ can be considered as a *probability measure defined on the Lindenbaum–Tarski algebra* corresponding to $\mathcal{F}$ (see Section 4.1). Property (3.8) can also be defined for this algebra and probability.

There are many procedures for *constructing probabilities* on $\mathcal{F}$: to *take the average* of probabilities defined on different models (by integral), to *extend* probabilities from a subset of $\mathcal{F}$ (e.g. from the quantified formulas), to *transform* probabilities to $\mathcal{F}$ from models, etc.

We note that Scott and Krauss introduced *σ-additive probabilities* for *infinitary logics* (see [142]). The above properties and definitions can be generalized to that case as well.

3.6.2 Connections with the probability theory

Although *probability logic* and *probability theory* are concerned with essentially the same kind of problems, the emphasis they put on these problems as well as the approach to solving them are radically different.

Recall that in *Kolmogorov's well-known probability theory* the probability is considered as a probability measure defined on a Boolean set algebra.

From a logical point of view, this set algebra (the *algebra of events*) interprets the language of propositional logic (in fact, in probability theory *infinitary propositional logic* is used because probability theory utilizies σ-algebras, but we can safely confine our attention to ordinary propositional logic without loss of generality; see [44].). Thus, Kolomogorov's probability theory is based on propositional logic: for example, the concept of *conditional probability* may be seen as the probability associated with the entailment $\alpha \models \beta$ in propositional logic.

In view of the foregoing discussion, we may raise the following question: Are there any direct applications of probability logic in probability theory, or an influence exerted on probability theory by probability logic?

We may answer this question by pointing out that by the use of certain logical techniques and results (see Chapter 4), we can generalize the notion of probability from the level of structures of propositional logic (i.e., from Boolean set algebras) to those related to first-order logic (e.g., to algebras corresponding to first-order models or Lindenbaum–Tarski algebras).To see this, let us consider the following illustration.

First-order logic finds its main applications in the theory of *stochastic processes* in probability theory. For example, some *first-order properties* of a realization $X(t)$ of a real stochastic process are:

monotonicity of $X(t)$: $\forall t \forall s(t < s \to X(t) \leq X(s))$,

boundedness of $X(t) : \exists s \forall t(X(t) \leq s)$,

$X(t)$ *takes* n *different values at most*:

$$\exists s_1 \ldots \exists s_n \forall t(X(t) = s_1 \vee \ldots \vee X(t) = s_n).$$

In probability theory, the *extension* of the usual probabilities to the sets representing the above mentioned properties requires non-traditional methods, because these sets proved to be *non-traditional* from the viewpoint of the traditional probability and measure theory. Various "roundabout methods" must be applied in classical probability theory in order to reduce this extension problem to traditional results (see the separability theorems due to Doob and others).

Analyzing this extension problem from the viewpoint of logic we can see that the point is the *first-order character* of these properties. This is why classical probability theory based on propositional logic can handle this problem only with difficulties. Using the concepts and theorems listed in the previous section an elegant solution can be given for this extension problem, but we do not go into details of the solution here.

Finally, we note that there are special theories of probability in which the logical relevance of theories are emphasized. Such a probability theory is the so-called *Logical probability theory* that was elaborated by Rudolf Carnap, one of the pioneers of probability logic, who considered probability theory as a kind of inductive logic.

References to Chapter 3 are, for example: [11], [13], [154], [156], [145] [160], [142], [39], [58], [136], [91].

Chapter 4

LOGIC AND ALGEBRA

It is a historical fact that modern logic was developed in an algebraic form in the 19th century (Boolean, De Morgan, Pierce, etc.). The intensive research of the connection between algebra and logic was initiated in the first half of the 20th century by Tarski and his school. It turned out that the logic–algebra connection is a deep and important relationship in mathematics. The branch of mathematics investigating this connection is called *Algebraic Logic*. This connection is nicely illustrated by the so-called characterization theorems in which varieties and their algebraic properties are characterized by the formal properties of the axioms defining the variety (e.g. such a theorem is the one according to which: a class of algebra is closed under homomorphisms if and only if its axioms are positive sentences).

Algebraic logic can be *applied* everywhere, where logic is applied at all. The logic–algebra connection often allows us to make a more elegant and simple treatment of applications than merely using logic, e.g., the algebraic theory of specification and verification, algebraic semantics of programming languages, etc. "Logical algebras" as Boolean, cylindric and relation algebras play an important role in database theory, logic programming, logic design, the theory of automata, artificial intelligence.

There are two directions in the research of the logic–algebra relationship: starting from logic and starting from algebra. Here we will mainly concentrate on the first one. The basic idea of this approach is the following: *If a problem is given in logic, we try to translate it to the language of algebra. Then, we try to solve it in algebraic form using algebraic techniques* (in doing so, we often need universal algebra). *If the problem is solved we translate the solution back to logic.*

Recall that we distinguished between two different facets of logic: that of semantics and that of proof theory. In accordance with this distinction, we distinguish the algebraization of semantics and the algebraization of proof theory. By virtue of a logic having completeness, we can conclude that the two algebraizations coincide.

Any algebraization can be characterized by using purely logical concepts or by using purely algebraic ones. This will be yet another aspect to be considered when dealing with algebraizations.

The question, whether there is an "algebraization" for a given logic can be answered by "yes" for most well-known logics under appropriate conditions, but

this may not be true of particular logics (such as the modal logics S1,S2,S3). We do not give details here concerning the conditions needed for algebraization. A collection of general theorems shows that if a logic is algebraizable, then with the logical properties definite algebraic properties can be associated and vice versa. For example, such a theorem is the following: a logic has a kind of (strong) compactness if and only if its algebraization is closed under ultraproducts.

There are famous algebraizations for *propositional logic* (Boolean algebras), for *first-order logic* (cylindric algebras, polyadic algebras), for *arrow logics* (certain relation algebras), for *intuitionistic logic* (Heyting algebras), among others. We give here the details concerning the algebraization of propositional logic and only sketch that of first-order logic and arrow logic.

4.1 Logic and Boolean algebras

Now, as a generic example, we discuss the classical connection of *propositional logic* and algebra (more precisely, that of propositional logic and Boolean algebra). First we assume that the reader is familiar with the concepts of Boolean algebra.

We introduce an important class of Boolean algebras. Let U be an arbitrary set.

4.1 Definition *A field of sets (or a Boolean set algebra) with unit U is a structure*

$$\mathcal{H} = \langle H, \cup, \cap, \sim_U, U, \emptyset \rangle$$

where H is the universe consisting of certain subsets of U, and H is closed under the set operations $\cup$, $\cap$, $\sim_U$ and contains the sets U and $\emptyset$.

So the similarity type of a field of sets is $\langle 2, 2, 1, 0, 0; \rangle$.

Let $\mathcal{L}$ be the language of propositional logic, Σ be a theory in $\mathcal{L}$ (Σ is defined in a semantic way, so Σ is closed for $\models$) and $\mathcal{M}_\Sigma$ be such a *set* of Σ-models that for every model M satisfying all the formulas in Σ, a model $M' \in \mathcal{M}_\Sigma$ can be found such that M is elementarily equivalent to M'. For a given formula α, let $\mathcal{M}(\alpha)$ denote the set of models in $\mathcal{M}_\Sigma$ satisfying the formula α.

It is easy to see that $\mathcal{M}(a)'$s form a field of sets with universe $\mathcal{M}_\Sigma$. This field of sets is denoted by $\mathcal{E}^\Sigma$ and is called the field of *elementary classes*.

4.2 Definition The algebraization *of propositional logic based on semantics is the class $I(\mathcal{E}^\Sigma)$ of algebras, where I denotes isomorphic closure and Σ varies over the possible theories on the possible propositional languages.*

The algebraization of a *concrete logic* is the concrete algebra $I(\mathcal{E}^\Sigma)$ with the fixed models of the fixed theory Σ.

Recall that a class of algebras is called a *variety* if it can be axiomatizable by a set of equations.

4.3 THEOREM (algebraization of propositional logic)

(i) *the algebraization* $I(\mathcal{E}^\Sigma)$ *of propositional logic coincides with the isomorphic closure of the class of field of sets.*

(ii) $I(\mathcal{E}^\Sigma)$ *is a variety and it is the class of Boolean algebras.*

By *(ii)*, $I(\mathcal{E}^\Sigma)$ is *axiomatizable*, moreover, it is finitely axiomatizable (by the Boolean axioms). Thus part (ii) is like a Completeness Theorem in Logic, because a kind of semantics (*an "algebraic semantics"*) is characterized by axioms (by Boolean axioms).

By the famous *representation theorem* of Stone (namely, that every Boolean algebra can be represented as a field of sets) the propositions *(i)* and *(ii)* are equivalent.

Until now our discussion of the propositional logic was based on logical *semantics*. But it is possible to base it on *proof theory*, too.

Let us fix a *proof system*, say, a Hilbert type system, and a consistent theory Σ. We introduce an equivalence relation $\sim$ defined on the formulas in $\mathcal{L}$ by

$$\alpha \sim \beta \text{ if and only if } \Sigma \vdash \alpha \leftrightarrow \beta.$$

Let $\|\alpha\|$ denote the equivalence class containing the formula α. We can define the Boolean operations and constants 0 and 1 for the objects $\|\alpha\|$:

$$\|\alpha\| + \|\beta\| = \|\alpha \vee \beta\| \ , \quad \|\alpha\| \cdot \|\beta\| = \|\alpha \wedge \beta\| \, , \quad -\|\alpha\| = \|\neg\alpha\| \, ,$$
$$0 = \|\alpha \wedge \neg\alpha\| \, , \quad 1 = \|\alpha \vee \neg\alpha\| \, .$$

We can prove that we obtain a Boolean algebra in this way (it is denoted by $\mathcal{B}^\Sigma$ *and called the* Lindenbaum–Tarski algebra *corresponding to the language* $\mathcal{L}$ *and theory* Σ*).*$\Sigma = \emptyset$ is an important special case: it can be seen that $\mathcal{B}^\emptyset$ is a so-called "*absolutely free*" algebra. We will denote $\mathcal{B}^\emptyset$ by Fr below.

4.4 DEFINITION The algebraization of propositional logic based on a given proof system *is the class* $I(\mathcal{B}^\Sigma)$ *of algebras, where* Σ *varies over the possible theories on the possible propositional languages.*

The following proposition is true:

4.5 THEOREM $\mathcal{E}^\Sigma$ *and* $\mathcal{B}^\Sigma$ *are isomorphic algebras* ($\mathcal{E}^\Sigma \simeq \mathcal{B}^\Sigma$).

This isomorphism corresponds to a completeness theorem for propositional logic (the proof also makes use of Completeness theorem of Logic), in particular, to the completeness of the *given* (Hilbert style) proof system. A Corollary is that Theorem 4.3 is true for the algebraization $\mathcal{B}^\Sigma$ besides $\mathcal{E}^\Sigma$. Therefore, the

algebraizations $I(\mathcal{E}^\Sigma)$ and $I(\mathcal{B}^\Sigma)$ *coincide* (they are exactly the class of Boolean algebras).

Summing up the algebraizations above, with *propositional logic using semantics and proof theory we can associate directly the classes* $I(\mathcal{E}^\Sigma)$ *and* $I(\mathcal{B}^\Sigma)$ respectively, *and indirectly the concept of* I*(set of fields) and a "class of abstract Boolean algebras"*. All of these classes coincide.

Now we are in the position to formulate in an exact way the statement mentioned in Section 2.2.1 that the meaning (interpretation) functions of propositional logic (propositional models) can be considered as homomorphisms to the Boolean algebra of two elements.

4.6 THEOREM *The meaning function* m *defined in Definition 2.13 can be considered as a* homomorphism *from* Fr *to the Boolean algebra of two elements.*

We note that formula algebras can be introduced only using semantics. Let us introduce an equivalence relation $\equiv$ defined on the formulas in $\mathcal{L}$:

$$\alpha \equiv \beta \text{ if and only if } \mathcal{M}(\alpha) = \mathcal{M}(\beta).$$

Let $\|\alpha\|'$ be the equivalence class containing α. We can define the Boolean operations $+,\ \cdot,\ -$ and constants 0 and 1 for the objects $\|\alpha\|'$. By the completeness theorem, the algebras obtained in this way are also isomorphic to Lindenbaum–Tarski algebras.

We can make a "*dictionary*" (table) how to assign Boolean algebraic concepts to concepts of propositional logic, and vice versa.

Boolean algebras	**Propositional logic**
$+, \cdot, -$, relation $\leq$	$\vee,\ \wedge,\ \neg$, relation $\vDash$
Boolean filter	logical theory
Boolean ultrafilter	complete logical theory
Boolean constant 1	set of tautologies
Boolean ultrafilter theorem	Lindenbaum theorem
homomorphism from Fr into $\mathcal{D}$	propositional model
Boolean algebras are a variety	propositional logic has completeness

Recall that Lindenbaum's theorem states that every consistent theory can be extended to a complete and consistent theory.

Until now, we have started out from logic (propositional logic) and investigated how to assign an algebraization to logic. But what is about the opposite procedure, *what kinds of usual algebraic objects define a logic and how?*

An example of this procedure is the definition of propositional semantics, based on algebra. We use a kind of inverse of Theorem 4.6. With an absolutely free Boolean algebra Fr and a set h_i $(i \in I)$ of homomorphisms from Fr to the Boolean algebra of two elements $\mathcal{D}$ we can associate the formulas of a

propositional language and a set of propositional models. Therefore, the pair $\langle Fr, \{h_i\}_{i\in I}\rangle$ can be considered as a kind of propositional semantics.

4.2 Algebraization of first-order logic

We are going to define the algebraization of first-order logic as an extension of that of propositional logic. We have to find the algebraic counterpart of the logical connectives of first-order logic not included in propositional logic: $\exists$, $=$ and the individuum variables $x_1, x_2, \ldots$ (quantifier $\forall$ can be expressed by $\exists$ and $\neg$).

The main idea is that the constants $\exists$ and $=$ will be modelled *together* with those individuum variables to which they are applied. For each quantification $\exists x_i$, an operation (the so-called i-th *cylindrification*) will be introduced and for each equality $x_i = x_j$, $i = 1, 2, \ldots$, $j = 1, 2, \ldots$ a 0-ary operation will be introduced (this called ij-th *diagonal*). That is, *infinitely many operations are considered*.

Now we define the algebraization of the concept of "first-order model". The starting point is that the "truth sets" associated with a model M form a Boolean set algebra $\mathcal{K}$ (field of sets). This Boolean structure reflects point (ii) of the definition of "truth set" concerning propositional operations but *does not* reflect point (iii) concerning quantifiers and the presence of equality. $\mathcal{K}$ *must* be extended by the set operations "cylindrifications" and by the constants "diagonals". Now the definition is:

4.7 Definition A cylindric set algebra $\mathcal{B}$ *of* ω *dimension is a special field of sets* $\langle B, \cup, \cap, \sim_U, U^\omega, \emptyset\rangle$, *extended by an infinite collection of unary operations* C_i, $i \in \omega$ *(the* i*-th cylindrifications) and by an infinite collection of constants* D_{ij}, $i, j \in \omega$ *(the* ij*-th diagonal elements):*

$$\mathcal{B} = \langle B, \cup, \cap, \sim_U, U^\omega, \emptyset, C_i, D_{ij}\rangle_{i,j\in\omega}$$

where C_i *is such that for every* $a \in B$

$$C_i a = \left\{x \mid x^i_u \in a, x \in U^\omega, u \in U\right\}$$

(x^i_u *is obtained from* x *by replacing the* i*-th member of* x *by* u*) and*

$$D_{ij} = \{x \mid x_i = x_j, x \in U^\omega\}$$

further the universe B *is assumed to be* closed *under all the operations in* $\mathcal{B}$.

Geometrically, $C_i a$ means "forming a cylinder set" (see in Section 2.1.2) parallel to the i-th axis with respect to the element $a \in U^\omega$ and the constant D_{ij} means the ij-th "bisector" as a hyperplane. From the viewpoint of *logic* C_i and D_{ij} correspond to forming an existential quantification with respect to the variable x_i, and correspond to the equality $x_i = x_j$, respectively.

We can check that with a *first-order model* (with universe U) we can associate the *cylindric set algebra* (with unit U^ω) such that the elements of this algebra are the truth sets of formulas in the language (i.e. relations defined on U). This

is why a cylindric set algebra is also-called an "algebra of relations" (but *it is not* the usual relation algebra).

An immediate generalization of the concept of cylindric set algebra is the "*generalized cylindric set algebra*". Here, the unit U in the above definition of cylindric set algebra is assumed to be of the form $\bigcup_{k\in K} U_k$, where $U_i \cap U_j = \emptyset$ if $i \neq j$, $i, j \in K$. With a set of first order models with disjoint universes, a generalized cylindric set algebra can be associated.

This concept has a similar role in the first order case as the role of field of sets in the propositional case. In contrary to fields of sets the class of generalized cylindric set algebras is *not* finitely axiomatizable, moreover it is not axiomatizable with a *finite* scheme. This is one of the most important differences between the algebraization of propositional logics and that of first-order logics.

A concept analogous to Boolean algebras in the first-order case is the concept of (abstract) "*cylindric algebra*" (not defined here).

We can generalize the algebras $\mathcal{E}^\Sigma$ and $\mathcal{B}^\Sigma$ defined in the previous section to obtain special cylindric algebras. These extensions correspond to first-order semantics and a concrete calculus, respectively.

Using these concepts the algebraization of first-order logic can be introduced in a similar way as that of propositional logic. Of course, the algebraization of propositional logic and that of first-order logic have many common and different features. The situation in the first order case is *more complicated* than the propositional one. For example, with first order logic, only particular ("locally finite") cylindric algebras can be associated, these kinds of algebras cannot be defined by a first order schema.

Further important algebraizations of first order logic are the *polyadic algebras* and *quasi-polyadic algebras*, due to Halmos ([92]).

Further important kind of algebraization is the class of *relation algebras*. A relation algebra is a direct algebraization of "*the logic of relation algebras*" (see Section 3.4.1). The status of relation *set* algebras is similar to that of "cylindric set algebras". Beside relation set algebras, abstract relation algebras can be defined, too. It can be shown that relation algebras are closely related to a first-order logic with three variables. Relation algebras can be considered as an *algebraization of first-order logic*, too, but they do not characterize first-order logic in a direct way. The advantage of this algebraization is that only *finitely many* operations are used, and, further, relation algebras are not of infinite dimensional (as opposed to ω-dimensional cylindric algebras).

References to Chapter 4 are, for example: [126], [96], [23], [92], [97], [60], [52], [9]

Chapter 5

LOGIC in COMPUTER SCIENCE

The role of logic in computer science is twofold:

One role is that logic helps the foundation of several fields of computer science. Its main reason is that logic is the only discipline which studies *meaning* in a strict formal way (that is, in mathematical way). This is why logic today is widely recognized to be one of the fundamental discipline of computing and problem solving (maybe the most important one).

The other role of logic in information systems is that logic provides not only a theory, but the languages and calculi of logics can be transformed into *direct tools* of software development technologies (VDM, Z, HOL, TLA+ and so on), database systems and knowledge representation. Special logics can even be used as programming languages (like PROLOG, DATALOG).

We intend to show how logics help solving problems in some field of computer science - we, unfortunately, cannot provide a complete review in this book, but there are handbooks that we suggest the interested reader should consult (see the references). In this book only some interesting classes of problems are discussed such as

- how formal semantics can be provided for programming languages?
- how to ensure that programs (or other dynamic systems) behaves as it is intended (how to verify them)?
- how to represent knowledge in a formal way?

However no complete study can be given even for the selected topics - only the basic notions, some interesting aspects and approaches are discussed. Similarly, for lack of space, we cannot go into the details of some important areas of applications as database theory, functional programming, logic design etc.

5.1 Logic and Complexity theory

There is a close traditional connection between Logic and Algorithm theory, Logic and Automaton theory and between Logic and Complexity theory (e.g.

there are elegant proofs for Incompleteness theorems, undecidability of first order logic using the concept of Turing machine). Here we sketch some results mainly on the connections of Logic and Complexity theory.

We assume a minimal knowledge on basic concepts of computability and complexity theory: the concepts of the most known complexity classes **P** and **NP**, that of completeness with respect to a given complexity class, the reducibility of a problem to another one, etc. We accept here the hypothesis $\mathbf{P} \neq \mathbf{NP}$, as usual.

There are classical results in Complexity theory which are related to Logic. These are connected with important logical problems which are complete with respect to a given complexity class. Such a logical problem is the problem called SAT, that is, the problem of satisfiability of a propositional formula given in conjunctive normal form. As it is known, SAT *is a **NP**-complete problem*. A similar problem is the problem HORNSAT, that is, the problem of satisfiability of a finite set of propositional Horn clauses. It is known that HORNSAT *is a **P**-complete problem.* Other similar problems originated from logic are: the problems 3SAT, Max-2SAT, etc..

Problems above as complete problems are characteristics of a given complexity class. These kinds of results are important also from the viewpoint of applications. For example, the fact that HORNSAT is in **P** is the theoretical basis of logic programming (PROLOG is based on SLD algorithm concerning problem HORNSAT).

There is a less traditional connection between Logic and Complexity theory which is more direct and comprehensive. This connection is a bit similar to the logic-algebra connection. This is a kind of *translation from complexity theory to logic* and vice-verse. More exactly: *given a complexity class **C** we try to find a logical language $\mathcal{S}$ and a set Σ of formulas in $\mathcal{S}$ such that with every problem Q (or with a problem equivalent to Q) in **C**, we can associate a formula in Σ which formalizes Q. And conversely, with any formula in Σ one can associate a problem in **C**.*

The language of first order logic is *not suitable* for choice of $\mathcal{S}$, some extensions are needed. The main reason is that the concept of *recursion* is not expressible in terms of a first-order language. *Second-order* logic and its language seems to be sufficient. This is confirmed by the following theorem: *A language $\mathcal{S}$ is "recognized" by an automaton if and only if $\mathcal{S}$ is "definable" in (monadic) second order logic.*

The language $\mathcal{S}$ chosen for the characterization of complexity classes is the *second order language of graphs.* The class of models is restricted to the class of *finite graphs*. $\mathcal{S}$ contains only the binary relation symbol E, interpreted as "to be an edge" of the graph. Only relation variables (and no function variables) are used: one n-ary relation variable P_n for every natural number n.

The class of formulas of the form $\exists P_i \psi(P_i, x_1, \dots x_k)$ is considered, where P_i is the only relation variable, and $x_1, \dots x_k$ are the free individuum variables in ψ.

5.1 Definition ($\exists P\psi$ graph problem) *"The $\exists P\psi$ graph problem": Is the formula $\exists P_i \psi(P_i, x_1, \dots x_k)$ true for a given finite graph $\mathcal{G} = \langle G, E^{\mathcal{G}} \rangle$ and evaluation $\langle a_1, \dots a_k \rangle \in G^k$?*

For example, let us see the property "unreachability" in graphs that is the property "there is no path between given two nodes". A possible formalization of this property is

$$\exists P(\rho \wedge \neg P(x,y))$$

where $\rho = \forall u \forall v \forall w(Puu \wedge (Euv \rightarrow Puv) \wedge ((Puv \wedge Pvw) \rightarrow Puw))$. Here P should be a reflexive, transitive relation containing E. The formula says that there is such a reflexive, transitive closure of E for which $P(x,y)$ is *not* satisfied.

5.2 Theorem (Fagin) *Any $\exists P\psi$ graph problem is in **NP** and conversely, every decision problem of graphs which is in **NP** can be reduced to a $\exists P\psi$ graph problem.*

A related result is *Courcelle's* celebrated theorem: *Each problem definable in monadic second-order logic with additional predicates evaluating the size of sets, can be solved in linear time on graphs with bounded tree width.* This general and strong result allows many applications.

The next question is: how to characterize the problems in **P** inside **NP**, using logic.

Recall that a prenex normal form is a strong *Skolem form* if its prefix contains only universal quantifiers (it is an universal formula) and the kernel is of conjunctive normal form. A Skolem form is a *Horn formula* if in any conjunction of the kernel there is *at most one* literal without negation.

5.3 Definition (second order Horn formula) *A formula $\exists P\psi \in \Sigma$ is a second order* Horn formula *if ψ is a Horn formula with respect to P (the members of the conjunctions in the kernel of ψ are allowed to contain P at most once without negation).*

For example the conjunction members Puu, $\neg Euv \vee Puv$, $\neg Puv \vee \neg Pvw \vee Puw$, $\neg Pxy$ of the Skolem form of formula $\exists P(\rho \wedge \neg P(x,y))$ satisfy this condition, therefore this Skolem form is a second-order Horn formula.

It is not difficult to check that *if $\exists P\psi$ is a second order Horn formula then the $\exists P\psi$ graph problem is in **P*** (for example the problem "unreachability"). Unfortunately, the converse of this proposition *fails to be true*, a modification is needed.

Let us consider *linear ordered* graphs rather than "ordinary" graphs. The ordering is defined for the vertices of the graph. Linearity means that the ordering is irreflexive, trichotome and transitive. Let us extend the language S by a binary relation symbol R which is always interpreted as a linear ordering.

We can define a new problem called *$\exists P\psi$ ordered graph problem* similarly to the *$\exists P\psi$ graph problem* but restricting the graphs (the models) in the original definition to *linear ordered graphs*.

Intuitively, with an ordering of a graph we can associate a sequence of the elements of the graph. These elements, in the given order can serve as input for an automaton (e.g. a Turing machine).

The following theorem gives a logical characterization for the class **P:**

5.4 THEOREM (characterizing class **P**)

(i) *If $\exists P\psi$ is a Horn formula of second order then the $\exists P\psi$ - problem is in **P**.*

(ii) *Every decision problem of ordered graphs in **P** can be reduced to a $\exists P\psi$ ordered graph problem.*

Many other complexity classes can be characterized using logic. In order to do this different kinds of logics are needed. The so called "*fixed-point logics*" seem to be very adequate tool for this purpose. *Complexity classes* **P** ,PSPACE, LOGSPACE, NLOGSPACE *can be characterized as* the finitely axiomatizable classes in certain fixed-point logics.

Some remarks concerning *applications* and advantages of the connection between logic and complexity theory:

- This connection may help to recognize that a concrete problem is in a given complexity class (by expressing the problem in a suitable logical language), and it may provide us with better understanding of a given complexity class using the given logic.

- Logical characterizations allow us to convert problems, methods and results of complexity theory into logic and vice versa (for example, the statements LOGSPACE$\neq$SPACE or **P**=**NP** can be converted to logical statements and they can be investigated in logic, too).

- This connection allow us to consider a logical language associated with a given complexity class also as a programming language of higher level. Conversely, given a programming language, considering it as a logical language we may try to find a complexity class associated with this language. These kind of connections are actively investigated for different programming languages, among others for some kinds of the language DATALOG.

While we know that existential second-order logic precisely captures NP on the class of all finite structures, most other logical charatcterizations known up-to-date are restricted to particular classes of finite structures.

When considering the *invariant* versions of the above logics (e.g., order-invariant formulas, arithmetic-invariant formulas, etc.), one obtains charatcterizations of the according complexity classes with respect to the class of all finite structures. However, these logical systems *do not have a decidable syntax*.

In the last few years, a lot of research has been devoted to identifying larger and larger classes of finite graphs for which decidable characterizations of precisely the graph properties computable in polynomial time can be found.

In the literature, several computation models that operate directly on structures (rather than on their string encodings, as Turing machines or random

access machines) have been proposed; the one of the most prominent is Gurevich's *abstract state machines*. Based on the latter, Blass, Gurevich, and Shelah introduced two complexity classes for *choiceless* polynomial time computations with or without counting facilities, called CPT+C and CPT, respectively. The according machine models operate directly on structures in a manner that *preserves symmetry* at every step of a computation. The computations are called "choiceless", because they disallow choices that violate the inherent symmetry of the input structure. This inability of making choices is compensated, among others, by *parallelism* (i.e., the power to explore all choices in parallel).

References to Section 5.2 are, for example: [129], [31], [79], [112], [111], [1], [27].

5.2 Program verification and specification

5.2.1 General introduction

Software industry has a very dubious character, since there is no reliable quality control for its products. This was the origin of the so called "software crisis" at the beginnings of sixties. Since then software engineering has developed several software technologies to overcome the crisis.

The idea to use logic for the verification of programs exists parallel to the problem of quality control. To discuss the different logic based approaches first the *meanings of programs* have to be clarified and to overview the *program properties* that can be investigated by logics.

Data types are interpreted in relational structures (e.g. in many sorted models). The so called *state transition semantics* is considered. The simplest notion of a *state* is an evaluation of program variables. Statements of the programming language as well as programs are considered *state transitions*. A state transition is defined as the set of pairs of input and output states. So the *meaning* of a program is the set of pairs $\langle q_i, q_o \rangle$ such that there is an execution with output q_o for input q_i. This set is called the *input-output relation* of the program. Clearly the input-output relation is a partial function in the case of deterministic programs. The sequence of states of an execution - called *trace* - is sometimes considered as the meaning of the program.

The most apparent program properties are *correctness* and *termination*, however, the concept of correctness has to be clarified. Correctness can be defined with respect to (w.r.t.) a *specification* expressed by a pair of formulas $\langle \varphi(\overline{x}), \psi(\overline{x}) \rangle$. Formula $\varphi(\overline{x})$, called *precondition*, constraints the input state of the program, and $\psi(\overline{x})$, called *postcondition*, specifies the output state. Notice that while these two conditions contain the same variables, they refer to different evaluations. While a pair of pre- and postconditions may specify the program execution partially, the *weakest precondition* and *strongest postcondition* can be considered as the total specification of the program.

Let $\mathbf{p}$ be a (non deterministic) program, φ and ψ *be* first order formulas. The following definitions give the meaning of some generally used program properties.

5.5 Definition (safety - invariance - properties) *The* safety (invariance) properties *express that a* "good property" is always true. *Some of the important ones:*

- "a program p is partially correct w.r.t.$\langle\varphi(\overline{x}),\psi(\overline{x})\rangle$" *means that* "if the input state satisfies condition $\varphi(\overline{x})$, each output state of p satisfies condition $\psi(\overline{x})$";
- "program p has invariant $\psi(\overline{x})$ at a point of the program w.r.t precondition $\varphi(\overline{x})$" *means that* "if the input state satisfies condition $\varphi(\overline{x})$,condition $\psi(\overline{x})$ holds when any execution of p reaches the point marked by the invariant".

5.6 Definition (liveliness - eventually - properties) The liveliness (eventually) properties *express that a* "good property" eventually happens. *Some of the important ones:*

- "program p quasi-terminates w.r.t precondition $\varphi(\overline{x})$" *means that* "if the input satisfies condition $\varphi(\overline{x})$, program p has a terminating execution path";
- "program p terminates w.r.t precondition $\varphi(\overline{x})$" *means that* "if the input satisfies condition $\varphi(\overline{x})$, every execution path of program p terminates";
- "program p is quasi-totally correct w.r.t. $\langle\varphi(\overline{x}),\psi(\overline{x})\rangle$" *means that* "if input satisfies condition $\varphi(\overline{x})$, program p has an output state satisfying condition $\psi(\overline{x})$";
- "program p is totally correct w.r.t. $\langle\varphi(\overline{x}),\psi(\overline{x})\rangle$" *means that* "if input satisfies condition $\varphi(\overline{x})$, every output state of program p satisfies condition $\psi(\overline{x})$".

The traditional notation for partial correctness is $\{\varphi\}\mathrm{p}\{\psi\}$, for total correctness is $[\varphi]\mathrm{p}[\psi]$.

Theoretical computer science usually considers some metalanguage as programming language. If it contains some kind of iteration, its precise form does not count: the theoretical results do not depend on the detailed syntax. In the definitions non-deterministic programs are supposed, which have well-defined starting point and termination. In the case of deterministic programs the notions "*quasi-terminating*" and "*terminating*" coincide; similarly, the notions "*quasi-totally correctness*" and "*totally correctness*" coincide. Notice that termination can be expressed in terms of total correctness with **true** as postcondition; further, looping can be expressed by partial correctness with postcondition **false**. In the case of a program which fulfills its function continuously (like operation systems, word processors etc.), the state of termination is not characteristic. In that cases the postcondition can be assigned to any point of the program.

Naturally, there are many other interesting program properties that can be investigated by logic. For example, whether a parallel (or cooperative) program

is deadlock-free. Recently, almost any program property has been discussed in terms of logic except for performance properties. As an example, we refer to [104], which studies the formal handling of "secure information flow". Nowadays the multi-agent technology raises the problem of spatially distributed computation. In this case we have to be able to express statements like "there is a process at location n, which is able to satisfy specification $\bar{\mathcal{A}}$". The interested reader may want to visit web-site research.microsoft.com for further information.

In the following, we outline the theoretical base of program verification. All the statements are independent of the precise syntax of the programming language; we only have to suppose that some form of iteration is included. In the traditional research "while language" is used. Variants of *state transition semantics* are considered as the semantics of the programming language.

First [63] proposed a method to prove partial correctness, his method was formalized as a calculus by "Hoare". The method basically is an induction on the structure of the program. The *Floyd-Hoare calculus* consists of inference rules corresponding to the program constructions. Let us see two examples, the introduction rules for the sequence of statements and for while statement.

$$\frac{\{\varphi\}\mathrm{p}_1\{\lambda\} \quad , \quad \{\lambda\}\mathrm{p}_2\{\psi\}}{\{\varphi\}\mathrm{p}_1;\mathrm{p}_2\{\psi\}};I \tag{5.1}$$

$$\frac{\varphi \wedge \neg\alpha \to \psi \quad , \quad \varphi \wedge \alpha \to \lambda \quad , \quad \{\lambda\}\mathrm{p}\{\lambda\} \quad , \quad \lambda \wedge \neg\alpha \to \psi}{\{\varphi\}\ \mathbf{while}\ \alpha\ \mathbf{do}\ \mathrm{p}\ \mathbf{od}\ \{\psi\}}\ while\ I \tag{5.2}$$

Note that in both cases an *invariant assertion* (denoted by λ) is used for the decomposition of the program construction in question. In the first case the invariant can be generated automatically; however, to find a suitable invariant for loops needs human intuition, that is, understanding the program.

5.2.2 Formal theories

To formalize computing has been a great challenge for logic. There are several approaches, let us list some of them:

- Relation algebras and its logics are widely used, since relations can be considered as meanings of programs.
- New logics are defined for modelling computing. New connectives are defined as in *linear logic* (see [81]), or semantics is modified as in *dynamic predicate logic*[1] (see e.g. [86]). The most widely known such logic is *dynamic logic*, which is discussed below.
- Non-classical logics are used e.g., *intensional logic* or different *temporal logics* (discussed below).

[1] It is interesting that the semantical approach behind this logic originates in modelling of computing; however, it is used in linguistics rather than in computing. In spite of the similar name, dynamic predicate logic and dynamic logic are two basically different logics.

- Even *first order classical logic* can be used - a version is discussed below.

Naturally, the most important problem is the *completeness* of the proof system to be used. The issue can be raised independently on the logic in question. It seems that no proof system proving partial correctness can be complete, since looping can be expressed as partial correctness - and it is known that termination is an undecidable problem. It is strange that it was not realized while methods to prove program properties were developed. The reason is that a special notion of completeness was investigated. The first results of non-completeness were developed in the late seventies, see [77] and [162]. There were several attempts to cope with it, e.g. the notion of arithmetical completeness was used to characterize computational logics. Arithmetical completeness compares a computational logic to the logic of arithmetics: it means that the logic in question would be complete if arithmetics was.

The real solution for the problem of completeness was given by the Hungarian school, namely in [10] it is created a complete computational logic, by modifying the notion of termination. This will be discussed in the subsection below about first order programming theory.

Definabilty may also play an important role in programming theories, however we do not have enough space to discuss it.

Dynamic logic

Dynamic logic was developed in the late seventies; [93] is a good summary.

Dynamic logic is a multi-modal logic, where modal operators are labelled by *programs*. It means that classical logics are extended by modal operators $[\mathbf{p}]$ as a box operator and $\langle \mathbf{p} \rangle$ as a diamond operator, where $\mathbf{p}$ denotes a program. Notice that programs are defined by a grammar too, so different programming languages have different dynamic logics. However, the properties of different dynamic logics do not differ. Generally, non-deterministic programming languages are considered, and a meta language armed with test statement and iteration is defined as programming language. Here we outline dynamic logics as it was defined by Harel and his co-workers, however now new logics are based on traditional dynamic logic, e.g., mu-calculus combines dynamic logic with the fixed-point operator.

In the semantics of dynamic logic the set of worlds is the set of states and the accessibility relation is represented by the executions of the program in question. For every world the same structure is rendered, which represents the data types of the programming language in question.

To get the definition of the meaning function of first order dynamic logic, that of classical first order logic has to be modified as follows.

5.7 Definition (meaning of dynamic modal operators)
Let $P_{\mathbf{p}}$ denote the input-output relation of $\mathbf{p}$. The truth set of $[\mathbf{p}]\,\alpha$ is the set:
$\{q \mid \langle q, a\rangle$ *belongs to* $P_{\mathbf{p}}$ *for* ***any*** *evaluation* $a \in [\alpha]\}$,
the truth set of $\langle \mathbf{p} \rangle\,\alpha$ is the set:
$\{q \mid \langle q, a\rangle$ *belongs to* $P_{\mathbf{p}}$ *for* ***some*** *evaluation* $a \in [\alpha]\}$.

Remember that $q \in [\alpha]$ denotes that q is included in the truth set of α.

Intuitively $[\mathbf{p}]\,\alpha$ says that α is true after *any* execution path of $\mathbf{p}$, and $\langle \mathbf{p} \rangle\,\alpha$ says that *there is* an execution of $\mathbf{p}$ such that α is true after it[2]. Note that he truth set of the formulae are the set of inputs. We defined first order dynamic logic; however, propositional one is important, too.

The important program properties can be expressed in dynamic logic as follows:

program $\mathbf{p}$ is partially correct w.r.t. $\langle \varphi, \psi \rangle$	$\varphi \to [\mathbf{p}]\psi$
program $\mathbf{p}$ is quasi-totally correct w.r.t. $\langle \varphi, \psi \rangle$	$\varphi \to \langle \mathbf{p} \rangle\,\psi$
program $\mathbf{p}$ quasi-terminates w.r.t. precondition φ	$\varphi \to \langle \mathbf{p} \rangle\,\mathbf{true}$.

For deterministic programs total correctness is the same as quasi-total correctness, however, for non-deterministic programs it is complicated to construct the formula expressing total correctness.

The inference rules (5.1) and (5.2) can be expressed by dynamic logic axioms:
$((\varphi \to [p_1]\lambda) \wedge (\lambda \to [p_2]\psi)) \to (\varphi \to [p_1; p_2]\psi)$,
$((\varphi \wedge \neg\alpha \to \psi) \wedge (\varphi \wedge \alpha \to \lambda) \wedge (\lambda \to [p]\lambda) \wedge (\lambda \wedge \neg\alpha \to \psi)) \to$
$(\varphi \to [\mathbf{while}\ \alpha\ \mathbf{do}\ p\ \mathbf{od}]\psi)$
respectively. Clearly these axioms can be expressed as inference rules, too.

5.8 Theorem (compactness and completeness)
Propositional as well as first order dynamic logics are not compact.
Propositional dynamic logic is weakly complete, but it is not strongly complete.
First order dynamic logic is not weakly complete.

The theorem above holds for the traditional interpretation of termination; however, the negative results can be turned into positive, as it will be shown in the discussion of first order logic with explicit time below.

Notice that dynamic logic is adequate also to model any processes if they can be decomposed into elementary subprocesses.

Temporal logic

Temporal logic was first used in [133] to characterize a method of proving program properties. Since then, temporal logic has been widely used, mainly for parallel programs.

Temporal logic applied to programming uses other temporal operators than the standard ones described in Section 3.2. The modalities referring to the past are superfluous. The time scale is a discrete ordering, representing the steps of the execution of a program. To handle such a timescale new operators are introduced. Different approaches may use different operators, e.g. [133] introduces modality "**until**", [79] uses modalities **1** with the meaning "it is true in the first moment" and "**next**" with the meaning "it is true in the next time moment".

While the syntax of dynamic logic contains the syntax of the programming language, the elements of the syntax of the programming language do not occur in the temporal logical description of the execution of a program. For a program

[2] Remember that non deterministic programs are being discussed.

a set of axioms is generated, which describe the possible traces. We do not have enough space here to detail how to perform it. Non rigid constants correspond to program variables, while rigid variables are used in the formulas describing properties of states. The history of a program variable can be obtained by a function that maps the time scale to the evaluations of a constant representing the program variable[3]. However, [112] adds "actions" to his time logic (TLA), which has been developed into a practical specifying language - below we discuss it. In TLA *actions* are introduced directly, it makes the descriptions of the program executions easier and more to intuitive. It is a peculiarity of the logic that actions are formulae at the same time: an action is true in a time structure if it transmits the first state into the second one.

In the propositional case temporal computational logic can be embedded into dynamic logic, that is, any temporal formula can be expressed by a dynamic formula. However [79] proves that the two first order logics cannot be compared: there is a temporal formula that cannot be expressed in dynamic logic, and a dynamic formula that cannot be expressed in temporal logic.

Traditionally, temporal logic uses the ordered set of natural numbers as the time scale, and therefore it is not complete in the first order case. [110] provides a comprehensive overview of the result of the traditional approach. If the time scale belongs to a more general class (like the class of sets with discrete ordering and successor and precedessor functions), a complete temporal logic can be developed. [12] provides a survey of temporal logics with general time scales used for programming theories.

First order logic with explicit time

First order logic with explicit time can be used to characterize program execution. It can be considered as a translation of temporal logic into first order one. It was published first in [80], a detailed discussion can be found in [79].

For the sake of simple presentation we use a many sorted logic. Beside data type sorts additional sorts are added:

- *time sort* (t), corresponding to the time scale of temporal logic,
- *function sorts*[4] (d^{t}) for any data sort d, representing the history of a program variable of sort d in the execution, so they stand for functions mapping time to data.

The language has to be supplemented with additional non-logical symbols to handle time (e.g. constant 0 for starting moment, function **s** with the meaning "next", and the predicate ¡). For any data sort d function symbol $\mathbf{app}_{\mathrm{d}}$ with arity $\langle t, d^{\mathrm{t}}; d\rangle$ is added to the language, that is, its interpretation in a model is a function that tells us what is the value of the function in a given moment. Let us denote the set of sentences of the completed language by F_t.

Axioms are needed to ensure that in every model the interpretations of the additional symbols reflect their intuitive meanings. These axioms belong in the following three groups, where t denotes a time variable, f and g denote function variables:

[3] ÷ In intensional logic such functions can be referred in the language, they form the "intensions" of the constants.

[4] Some authors use the terms *path* or *intension*.

- Axioms for time, describing a discrete ordering.
- Induction axiom scheme for time: $(\alpha(0) \wedge \forall x(\alpha(x) \to \alpha(\mathbf{s}(x)))) \to \forall x \alpha(x)$.
- Axioms for function sort:
  - axioms for equality: $\forall t\ \mathbf{app}_{\mathrm{d}}(f,t) = \mathbf{app}_{\mathrm{d}}(g,t) \to f = g$ for every data sort d;
  - different versions of comprehension scheme expressing that there are as many elements of function sort as we need, that is e.g. for every definable function there exists an element of the function sort: $\forall t \exists! y \alpha(t,y) \to \exists f \forall t \alpha(t, \mathbf{app}_{\mathrm{d}}(f,t))$.

Models satisfying these axioms are called *models with inner time*. Let us denote this class by M_t. A model is called a *standard model* if the time sort is isomorphic with natural numbers. Let M_t^s denote the class of standard models. Logic $\langle F_t, M_t^s, m\rangle$ is called *standard time logic*, $\langle F_t, M_t, m\rangle$ is called *time logic*.

In standard time logic the interpretation of termination is the traditional one: the program stops after finitely many steps. However, in time models the program may stop at a non standard moment. To characterize the traces in non-standard moments the description of the program has to be completed by an *induction axiom*, saying that

if "a property is true at the starting point", **and** "if this property is true in a moment, it remains true in the next moment"
then "this property is true in every time moment".

5.9 Theorem (completeness) *Standard time logic is not complete, time logic is strongly complete.*

In time logic it is clear why traditional programming logics are not complete. While data are considered as models of the axiom system in question, time is supposed to be a standard model, which cannot be axiomatized. The interpretation of termination spoils completeness even in the case of logics where time does not appear explicitly. If also non-standard time scales are considered, completeness is ensured. First [10] realized this. Induction axiom for traces can be expressed in dynamic logic as well as in temporal one, so complete versions of these logics can also be created - see [79].

Clearly, temporal logic can be transformed into time logic, this can be proved about dynamic logic, too.

5.2.3 Logic based software technologies

The analysis of programs by logical means first aimed to prove various properties of existing programs. However, the paradigm changed soon: the aim became *to write correct programs*, and logic based software development technologies have been developed to this aim. Techniques follow a top-down development strategy and may consist of

- a specification language, which supports to specify software products in several levels,

- methods to prove that refinements of the specifications are correct,
- methods to generate code for the specification.

Specification languages are generally based on logic. Note that pure logic cannot be used, because

- a specification has to be *structured*, that is, some kinds of units are needed, which can refer one to another - but logic studies only *sets* of sentences, no other structure;
- the type structure of programming language has to be represented in the logic used for specification;
- the specification speaks about higher order objects (e.g., functions) - constructions to define such objects have to be provided for the specification language.

[112] illustrates how a specification language is built up[5].

The earlier specification languages, which are studied and/or used in practice even now, are

- VDM (Vienna Definition Method), whose development started in the late seventies, the first detailed report is [28], and
- Z, which was developed through the 1980s, the first reference manual was published in 1989: [146].

[103] and [102] are thorough tutorials of VDM and Z respectively.

Stepwise refinement is the focus of these tools. The specification language helps the specification of the *data types* used in the system and the specification of *states* and *state transformations* (that is, specification of programs). The specification consists of two parts, the first is the definition of non logical symbols - this is done by set theoretical formalism. In the case of the definition of data types the second part consists of axioms, in the case of definition of program modules it consists of their specifications (usually in the form of pre- and postconditions).

The steps of program development consist of defining *refinements* of the existing definitions and proving the correctness of the refinements. The refinements may create new specifications in the same logics - in this case the correctness proof can be supported by one of the calculi for the logic used in the system (generally, the first order one). However, in several cases, the refinement interprets a data type of the higher level specification in another data type (e.g. sets are interpreted in sequences). In this case the transformation between logics has to be handled. At the end of the refinement process the specifications are transformed into programs - for this purpose special calculus is provided, similarly to the Floyd-Hoare calculus.

There are several studies comparing the two systems, like [27], [94]. They mostly conclude that there are neither important theoretical nor practical differences between the two systems. Naturally, the two systems differ in several

[5] The concrete language is TLA+, but the paper illustrates some general problems and solution.

points of view, and we think that the main difference is that relational algebraic operators - including operators for the specification of relational data bases - are built into Z. Both systems use naive set theory for specification, however [78] argues that the theory of hereditary finite sets would be more adequate for this purpose.

The most recent system following the same line is the *B method*, which is able to handle object oriented systems too - at least in some extent. To learn more about it visit www.b-core.com.

There are systems which combine specification with verification: a specification of a complex system can be described by them, and a proof procedure is provided to prove correctness:

- ESC/Java is a programming tool for finding errors in Java programs at compile time, which uses program verification techniques. See research.compaq.com/SRC/esc.
- TLA+, which is the extension of *time logic with actions*. "TLA BOOK" is a detailed introduction to the specification language, and it illustrates in details how to use such a specification language. For more information visit http://research.microsoft.com/users/lamport/tla/tla.html.
- Spin is a popular software tool that can be used for the formal verification of distributed software systems. The tool was developed at Bell Labs in the original Unix group of the Computing Sciences Research Center, starting in 1980. "Holzmann" presents techniques of using the SPIN model checker to verify correctness requirements for systems of concurrently executing processes. For more information visit spinroot.com/spin/whatispin.html.
- PVS is a verification system: that is, a specification language integrated with support tools and a theorem prover. The interested reader may visit pvs.csl.sri.com.
- HOL-Z is a proof environment for Z and HOL[6] specifications on the one hand and the generic theorem prover Isabelle on the other, see abacus.informatik.uni-freiburg.de/holz.

The systems outlined show that program verification is in the main stream of computer science, and it is suitable for practical applications. However, proving the correctness of systems demands huge resources, both in the computing and in human resources to create the specification of the system and to outline the plan of the proof. However, in critical safety systems program verification is applied. ",,," describes the use of TLA+ by an Intel processor design group.

References to Section 5.2 are, for example: [48], [71], [118].

[6] HOL stands for higher order logic.

5.3 Logic programming

The history of logic programming started with an application which was based on a resolution proof procedure, which used only so called Horn clauses and proved surprisingly effective. First it was thought that a logic program gives a specification of "**what**" the program is supposed to do that is that it specifies the input-output relation. However soon it was realized that using Horn clauses not only "**what**" to do is specified, but also "**how**" to do - at least to some extent. In other words: not only the input-output relation is specified, but an algorithm too. So there came a change of the paradigm in the application of theorem proving: *to use logic as a programming language*. A new slogan was announced by R. Kowalsky:

$$\text{ALGORITHM} = \text{LOGIC} + \text{CONTROL}.$$

The slogan expresses that the formulas of the logic programming language describe the essence of the algorithm, and the search strategy of the logic programming language fills the gaps. However the real merit of the slogan is that the segments of logic that are usable as a programming language, are ideal algorithm description languages.

For some years it was thought that the new paradigm can be extended to the whole first order logic, however, soon it was realized that the secret of the effectiveness is restricting the class of formulae. So even nowadays PROLOG and its relatives (including DATALOG) are the commonly used logic programming languages. One of the most interesting question of logic programming is *which segments of logics can be considered as programming languages*. This question has both theoretical and practical aspects.

A logic programming language is a proof procedure, that is, a proof system and a search strategy. An axiom system is considered a logic program, the proof of a theorem the execution of a logic program. Since logic programming originates in problem solving, it can be supposed that the *logic program consists of universal formulas* and *the theorem to be proved is an existential one*. The *theoretical condition* of "being a logic programming language" is that every proof has to provide terms for the existentially bound variables as *the result of the computation*; it is explained below in details.

Let Σ be a set of universal formulas, and let a theorem to be proved given in the form $\exists \overline{x} \alpha(\overline{x})$ (called goal or query). Recall that, according to Herbrand's theorem, $\Sigma \vDash \exists \overline{x} \alpha(\overline{x})$ is equivalent to $\Sigma \vDash \alpha(\overline{t_1}) \vee \ldots \vee \alpha(\overline{t_n})$ for some sequences $\overline{t_1}, \ldots \overline{t_n}$ of terms. But, it is important to ensure this equivalence for one sequence $\overline{t_i}$ only, that is to ensure the relation $\Sigma \vDash \alpha(\overline{t_i})$ for some i. The property $\Sigma \vDash \alpha(\overline{t_i})$ is called the "*existence of correct answer*" in the logic programming community. Clearly, the expectation for "*existence of correct answer*" is a strong restriction for the segment of logic used in logical programming.

The correct answer condition can be reformulated with the help of the provability relation too: $\Sigma \vDash \alpha(\overline{t_i})$ *if and only if* $\Sigma \vdash \alpha(\overline{t_i})$ *for a sequence* $\overline{t_i}$. The terms in $\overline{t_i}$ can be considered as the output of the "computation". Notice that this property one of the principles of *intuitionistic proof systems*.

It is more difficult to formulate precisely the practical aspects of the question that which proof procedures can be considered logic programming languages. On the one hand the search for the proof has to be effective, on the other hand

the "programmer" have to have the chance to foresee the process of search for the proof. Here lies the secret of the wide spread usage of PROLOG: *a logic program can be viewed and understood as a set of sentences of logic as well as a program, anticipating the execution*.

5.3.1 Programming with definite clauses

Here we sketch the logical background of the programming language PROLOG, and data base query language DATALOG, that is, *programming with definite clauses*.

Suppose that a first order language $\mathcal{L}$ is given without equality. Clauses will be meant here as first order clauses.

5.10 Definition (Horn clauses) *Clauses with at most one positive literal are called* Horn clauses. *Clauses with precisely one positive literal are called* definite Horn clauses *(in shortly definite clauses).*

Let the set Σ of axioms (Horn clause program) consist of *definite clauses*, where a definite clause may be written in the form $A_1 \wedge \ldots \wedge A_n \to B$, ($A_1, \ldots, A_n$, B are atomic formulas, $0 \leq n$). Notice that $n = 0$ is allowed, in this case the atomic formula in the implicative form is denoted by $\to B$.

Let the theorem (query, or goal) is $\exists \overline{x} \alpha(\overline{x})$ where α is of the form $A_1 \wedge \ldots \wedge A_n$ (it is denoted by $A_1 \wedge \ldots \wedge A_n \to$ by the logic programming community). The clause form (let's denote it β) of the *negation* of $\exists \overline{x} \alpha(\overline{x})$ is obviously a negative clause. The following theorem states that correct answer problem has a positive solution for these formulas.

5.11 Theorem (existence of correct answer) *If* $\Sigma \vDash \exists \overline{x} \alpha(\overline{x})$ *then* $\Sigma \vDash \alpha(\overline{x}/\overline{t})$ *for some sequence* $\overline{t}$ *of terms.*

The resolution rule that is used in PROLOG is called SLD resolution (S stands for selection function, L for linear resolution, D for definite clauses[7]).

5.12 Definition (SLD resolution rule) *Let* Q *be a negative clause in the form* $G_1 \wedge \ldots \wedge G_n \to$, *and let* S *be a definite clause in the form* $A_1 \wedge \ldots \wedge A_n \to B$. Q *and* S *can be resolved if and only if literals* G_1, *and* B *can be unified. Let* $\mu = mgu(G_1, B)$. $Res^{SLD}(Q, S) = A_1\mu \wedge \ldots \wedge A_n\mu \wedge G_2\mu \ldots \wedge G_n\mu \to$.

Notice the following properties of SLD:

- Two definite clauses are not allowed to be resolved.

[7] For general clauses one of the most effective refinement of resolution is the so called SL resolution (linear resolution with selection function). However the secret of the SL resolution is the elegant and effective way how it executes the resolution steps between two derived clauses - and this step does not occurs in the case of definite clauses, so in this case SL turns to be input resolution with selection function.

- The derived clauses are negative clauses, they are of the form of the goals.
- It is determined which literal of the negative clause has to be resolved This constraint is called using *selection function*, it saves completeness. In traditional PROLOG systems the clauses are stored as sequences of literals, therefore the first literal of the negative clause is selected for resolution. For this reason the ordering of literals in the derived clause is important.

5.13 Definition (SLD derivation) *Suppose that Λ is a set of definite clauses, and C is a negative Horn clause. The SLD derivation of a clause K is a sequence $\langle C_1, \ldots, C_k \rangle$ of negative Horn clauses such that $C_1 = C, \ldots, C_k = K$ and C_i is obtained as $Res^{SLD}(C_{i-1}, C_m)$ for some $C_m \in \Lambda$.*

Let Λ and C be as in the definition.

5.14 Theorem (Completeness of *SLD* resolution) *$\Lambda \cup \{C\}$ is unsatisfiable if and only if there is a derivation of the empty clause $\square$ by SLD resolution from the set $\Lambda \cup \{C\}$.*

There is a version of completeness that states the "*existence of correct answer*". Let Σ and $\exists \overline{x} \alpha(\overline{x})$ the same formula set and formula as in Theorem 5.11, and let Λ be the clause set corresponding to $S = \Sigma \cup \{\neg \alpha(\overline{x})\}$, and let us suppose that the empty clause can be derived from S by SLD resolution. Let σ_j^i be denote the most general unification applied to x_j in the i-th step of the derivation.

5.15 Corollary (existence of correct answer) *Let us consider terms $t_j = x_j \sigma_j^1 \ldots \sigma_j^n$, where n is the length of the derivation. Then $\overline{t} = \langle t_1, \ldots t_m \rangle$ is a correct answer, that is $\Sigma \models \alpha(\overline{t_i})$.*

For definite clauses a so called "*procedural semantics*" was also provided: a definite clause is a procedure with B as the *head* and $A_1 \wedge \ldots \wedge A_n$ as the *body*. Each A_i calls procedures whose head can be unified with A_i. In this interpretation "calling a procedure" corresponds to an execution of SLD resolution inference rule.

Let us see a simple example. In logic programs we often use data type *list* that is the language has a constant "nil" for empty list, and a binary function "." for list constructing operation. The logic programs act over the Herbrand universe generated by them that is they act over lists. The following simple program defines the "element of a list" relation (small case letters denote constants and upper case letters denote variables):

Element(X, L.X)←

Element(X, L.Y)←**Element**(X, L)

saying that "X is element of a list if it is the head of the list, or if it is the element the tail of the list".

Notice that the same program can be used for different aims. The goal **Element**(a,l) tests whether a is an element of l, **Element**(X,l) takes an element of l (the head - if the search strategy is considered too), **Element**(a,Y) creates a partially defined list with a as an element (as head).

If procedural semantics is considered then it can be said that a sophisticated calling mechanism is used for the parameters: the same parameter can be input or output one depending on the call of the procedure. It is called "call by need".

Further interesting properties of logic programming languages are as follows:

- there are no iteration in them, only recursive algorithms can be written
- there is no assignment statement, only bounding of variables.

Clearly the search space of SLD resolution can be considered a tree, where the branches are derivations. So the problem of the search for the answer (for the empty clause) is transparent. So the search for the solution is built into the interpreters. In PROLOG-like logic programming languages the depth first search strategy is used, therefore procedure calls (execution of resolution steps) can be *backtracked*.

These features were soon built into other programming languages too.

The most important features of the paradigm "*programming with Horn clauses*" can be summarized as follows:

1. The existence of a single intended model (the least Herbrand model) yields the semantics for Horn clause programs and ensures the existence of correct answer.
2. The Herbrand universe allows the programmer to use function symbols in the language as constructors of abstract data types, to encode objects as terms and to represent recursive definitions of these objects by means of Horn clauses.
3. Horn programs can specify any recursively enumerable set, so programming with Horn clauses captures the commonly accepted notion of computability.
4. Programming with Horn clauses is declarative due to the separation of logic, represented by sets of Horn clauses, from uniform control, represented by SLD resolution.

5.3.2 On definability

To understand the theoretical base of logic programming first we summarize some aspects of the theory of definability. Definability has an extensive theory in logic. We only discuss a special part of it, the theory of inductive definitions. This topic is particularly important in applications.

Let $\mathcal{L}$ be a first order language, and let R be a new k-ary relational symbol. Let $\mathcal{L}'$ be the language obtained from $\mathcal{L}$ by expanding it with R. The models of $\mathcal{L}'$ are denoted by $\langle M, R' \rangle$, where M is a model of $\mathcal{L}$ and R' is the interpretation of M.

We remark that only the case of one symbol R is discussed here for the sake of simplicity. But everything what is said in this section is true for the simultaneous introduction of more than one relational symbols.

5.16 DEFINITION (different forms of definitions) *A sentence of the form* $\forall\overline{x}(R(\overline{x}) \leftrightarrow \alpha(\overline{x}))$ *is called an* explicit definition *of* R*, if* $\alpha(\overline{x})$ *is a formula of* $\mathcal{L}$ *(that is, it does not contain* R*).*
A set of formulae Σ_R *consisting of formulae of* $\mathcal{L}'$ *including relational symbol* R *is called the* implicit definition *of* R*. If for a model* M *of* $\mathcal{L}$ *and relation* R' *it holds that* $\langle M, R'\rangle \vDash \Sigma_R$ *then it is said that* Σ_R *defines* R' *in* M*. If* R' *is the only interpretation of* R *such that it satisfies* Δ_R*, it is said that* Σ_R uniquely defines R' *in* M*.*
A sentence Δ_R *of the form* $\forall\overline{x}(R(\overline{x}) \leftrightarrow \alpha(\overline{x}, R))$ *is called the* inductive definition *of* R*, if* $\alpha(\overline{x}, R)$ *is a formula of* $\mathcal{L}'$*.* α *is called the* defining formula *of the definition*[8]*.*

The basic question of definability theory is that what kind of definition can uniquely define a relation. Clearly, an *explicit definition* uniquely define the relation in question, since the defining formula defines a well determined relation in every model of $\mathcal{L}$. As regards *implicite definitions* see the following theorem:

5.17 THEOREM (definability) *Suppose that an implicite definition* Σ_R *uniquely defines a relation on each model of* $\mathcal{L}$*. Then there is an explicit definition of the relation.*

Clearly, inductive definitions are "implicit definitions" of special form. The theorem implies that inductive definitions themselves are very weak concepts, this is why they are further analyzed below. If first order logic is considered, its expressive power is not greater than that of implicit definitions. However, in the theory of inductive definition strong results can be found (for the theory of inductive definability see [7]).

5.18 DEFINITION (least fixed point of an inductive definition on a model) *Let* M *be a model of* $\mathcal{L}$ *and* $\widehat{R}$ *be a relation defined by* $\Delta_R = \forall\overline{x}(R(\overline{x}) \leftrightarrow \alpha(\overline{x}, R))$*.* $\widehat{R}$ *is the* least fixpoint of Δ_R*, if for any* R' *defined by* Δ_R *it holds that*[9] $\subseteq R'$*.*

It can be proven that in general the least fixed point of an inductive definition *cannot be defined uniquely* in first order logic. The defining formula $\alpha(\overline{x}, R)$ can be considered as a set theoretically operator in algebraic logic. As regards the terminology "least fixed point" of an inductive definition, it is the same as the least fixed point of the set theoretical operator associated to the defining formula.

Some inductive definitions do not define relations at all (simplest example: $R(x) \leftrightarrow \neg R(x)$); or even if there are defined relations on M, there is no least one (simplest example: $R_1(x) \leftrightarrow \neg R_2(x)$, $R_2(x) \leftrightarrow \neg R_1(x)$).

[8] Note that if Δ_R defines R' in $\langle M, R'\rangle$, then the truth set of the defining formula in $\langle M, R'\rangle$ provides precisely R'.

[9] Relation $\widehat{R}$ is the least one in the set of defined relations according to the ordering $\subseteq$ (set theoretical inclusion).

5.19 Definition (PE definition) *If relation symbol R occurs only positively in a disjunctive normal form of the defining formula α, then the definition Δ_R is said to be* positive. *If the universal quantifier does not occur in some prenex form of α then the definition Δ_R is said to be* existential. *The positive existential definitions are called* PE definitions.

5.20 Theorem (existence of least fixed point) *For any model M positive definitions have a* least fixed point, *positive existential definitions have a* recursively enumerable *least fixed point.*

The construction providing the least fixed point for Δ_R in terms of a sequence of relations on M is defined as follows:

$R_0 = \varnothing$

$R_1 = \{q \mid \langle M, R_0\rangle \vDash \alpha(\overline{q})\}$

$\vdots$

$R_{n+1} = \{q \mid \langle M, R_n\rangle \vDash \alpha(\overline{q})\}$

5.21 Theorem (construction of least fixed point) *If Δ_R is a PE definition then $\cup_{n<\omega} R_n$ is the least fixed point of Δ_R in a model M.*

This construction has a more general form too: if *any set* of k-tuples is taken as R_0, the construction provides the least fixed point that includes R_0.

For positive definitions there exists a so called *greatest fixed point* too, and for positive universal definitions the complementer set of the greatest fixed point is recursively enumerable.

Inductive definability is applied in several fields of Computer Science, therefore some logics are defined with built-in fixed point operators, that is, in these logics there is a logical symbol, e.g. **fxd** for the least fixed point. In the definition of the syntax the condition "if α is a formula then **fxd**α is a formula" is inserted. The meaning of a formula **fxd**α is the least fixed point of the definition $\forall(R \leftrightarrow \alpha)$ where $\forall$ means universal closure. Clearly, this operator leads out of first order logic, since in several cases the least fixed point cannot be explicitly defined.

Here is an example: the least fixed point of the definition

$$R(x) \leftrightarrow (x = 0 \vee \exists y(x = s(y) \wedge R(y)))$$

is the set of natural numbers in every model of the Peano axioms, but it can not be defined in first order logic.

About fixed point logics see [49].

5.3.3 A general paradigm of logic programming

Since the meaning of a program is a relation, namely, the input-output relation, the general form of logic programs may be specified inspecting the definition of this relation. *The existence of the least fixed point ensures the fulfillment of the existence of a correct answer*, therefore positive inductive definitions are the adequate candidates for logic programs. Since the answer must also be computable, *positive existential (PE) definitions can be considered logic programs*. More precisely, a logic program is a set of PE definitions; however, every result for one definition holds for sets of definitions, too. *Any kind of calculus can be attached to the class of PE definitions*, the special syntax makes them quite similar.

It is shown below how we obtain a definite clause program from a PE definition if resolution is to be used. For resolution the definition has to be transformed into clausal form.

Let $\Delta_R = \forall\overline{x}(R(\overline{x}) \leftrightarrow \alpha(\overline{x}, R))$ be a PE definition. For the first step of the transformation the following theorem is needed:

5.22 THEOREM (about least fixed point) *There exists the least relation satisfying formula* $\forall\overline{x}(R(\overline{x}) \leftarrow \alpha(\overline{x\wedge}, R))$, *and it is the same as the least fixed point of* $\forall\overline{x}(R(\overline{x}) \leftrightarrow \alpha(\overline{x}, R))$.

So, if we are only interested in the least fixed point, Δ can be substituted by $\forall\overline{x}(R(\overline{x}) \leftarrow \alpha(\overline{x}, R))$. Let us execute the following transformations:

$\forall\overline{x}(R(\overline{x}) \leftarrow \alpha(\overline{x}, R))$
if and only if
$\forall\overline{x}(R(\overline{x}) \leftarrow \exists\overline{y}(\alpha_1(\overline{x}, \overline{y}, R) \vee \ldots \vee \alpha_n(\overline{x}, \overline{y}, R))$
where $\alpha_i(\overline{x}, \overline{y})$ is a conjunction for every $1 \leq i \leq n$
if and only if
$\forall\overline{x}\forall\overline{y}(R(\overline{x}) \leftarrow \alpha_1(\overline{x}, \overline{y}, R)) \vee \ldots \vee \alpha_n(\overline{x}, \overline{y}, R))$
if and only if
$\forall\overline{x}\forall\overline{y}(R(\overline{x}) \leftarrow \alpha_1(\overline{x}, \overline{y}, R)) \wedge$ ... $\wedge\forall\overline{x}\forall\overline{y}(R(\overline{x}) \leftarrow \alpha_n(\overline{x}, \overline{y}, R))$
which can be written as a set of Horn clauses
$\{\forall\overline{x}\forall\overline{y}(R(\overline{x}) \leftarrow \alpha_1(\overline{x}, \overline{y}, R)$, ...$\forall\overline{x}\forall\overline{y}(R(\overline{x}) \leftarrow \alpha_n(\overline{x}, \overline{y}, R))\}$.

Finally, if there are atoms of the form $x = t$ in a conjunction $\alpha_i(\overline{x}, \overline{y}, R)$, let us substitute x by t in the conjunction. In this way a definite Horn clause program is obtained.

Let us consider the program defining the "element of a list" relation, which was given above. It can be gained from the definition

$\forall$X,Y**Element**(X,Y)$\leftrightarrow \exists$T(Y=T.X) $\vee \exists$Z,T(Y=T.Z$\wedge$**Element**(X,T)).

The transformation into definite clause program:
$\forall$X,Y(**Element**(X,Y) $\leftarrow$ ($\exists$T(Y=T.X) $\vee \exists$Z,T(Y=T.Z$\wedge$**Element**(X,T)))) iff
$\forall$X,Y(**Element**(X,Y) $\leftarrow \exists$Z,T(Y=T.X $\vee$ (Y=T.Z$\wedge$**Element**(X,T)))) iff
$\forall$X,Y,T,Z(**Element**(X,Y) $\leftarrow$ (Y=T.X $\vee$ (Y=T.Z$\wedge$**Element**(X,T)))) iff
$\forall$X,Y,T(**Element**(X,Y)$\leftarrow$Y=T.X)$\wedge$
$\forall$X,Y,Z,T(**Element**(X,Y)$\leftarrow$ (Y=T.Z$\wedge$**Element**(X,Y)))

Written as Horn clause program:
Element(X,T.X),
Element(X,T.Z)←**Element**(X,T)

The selection of other calculi than SLD resolution, may result a logic programming language different from the languages based on resolution, see [152].

It is the user's free choice what (s)he considers a logic program: the implicative sentences (the sets of definite clauses) or the definition with equivalence (*completed definition* of the definite clause program).

5.3.4 Problems and trends

One of the most controversial issues of logic programming was the question of *negation*, which in fact includes two questions:

1. How to infer negative information from a definite program?
2. How negative literals can be used in logic programs?

If we want to answer the first question, first a decision has to be made: when can a negative literal be considered a consequence of a logic program? There are three alternative answers:

1. *If a tuple $\overline{q}$ does not belong to the least fixed point, then $\neg R(\overline{x})[\overline{q}]$ holds.* In this case the logic program is considered with implication instead of equivalence. Clearly if the least fixed point is not recursive, there is no complete calculus for negation. Sometimes this condition is called *closed world assumption* in the context of data bases.

2. *If a tuple $\overline{q}$ does not belong to the greatest fixed point, then $\neg R(\overline{x})[\overline{q}]$ holds.* This answer is adequate for the inductive definability approach. There is no complete calculus for this case, since the complementer set of the greatest fixed point of a PE definition may not be recursively enumerable. In this case the tuples can be partitioned into three sets: the least fixed point, the complementary set of the greatest fixed point, and the difference between the greatest and the least fixed points. This fact suggests that an adequate semantics can be defined based on three valued logic.

3. *A negative literal holds if it is the semantic consequence of the completed definition of the logic program.* This answer takes into consideration the whole class of models, therefore it is not adequate for the inductive definition approach. However, we have complete inference rule only for this solution: the so called *negation as failure*. Negation as failure was introduced for definite clause programs; it says that the negative literal holds if the positive cannot be proved from the logic program (set of Horn clauses). Negation as failure was criticized as a "non logical" method, however, soon its completeness was proved. Moreover, if Natural Deduction is applied to PE definitions to prove a negative literal (starting from the "$\neg$ introduction rule") the structure of the proof corresponds precisely to the negation as failure method.

Notice that an answer for the first question does not work for the second one. If negative literals occur in the defining formula, the existence of the least fixed point cannot be ensured. There were several trials to interpret a defined relation as the meaning of the non-positive definition (see e.g. [19]): the generally known approaches are the stratified programs, well founded semantics, which is the three valued version of stable model semantics [161]. Gergely and Szabó proposed an interesting kind of logic program, the so called *quasi equations*. A quasi equation consists of the separate definitions of the positive and the negative literals: $R(\overline{x}) \leftarrow \rho_1(\overline{x})$, $\neg R(\overline{x}) \leftarrow \rho_2(\overline{x})$.

In the last decades stable model semantics initiated a new paradigm of logic programming, which differs from PROLOG-like logic programming in several important aspects - therefore we discuss it in a more detailed way. If negated atoms are used in the body of a Horn clause, the logic program may not have a least model - only several minimal ones. The great change came when the *stable model semantics* was introduced: let us consider not one intended model, but a family of intended models. From this decision a new variant of logic programming - called *stable logic programming* (SLP, for short), or recently *answer set programming* - has been born.

If function symbols are used as in Horn logic programming (allowing to express any partial recursive function by its recursive definition) and several "intended" models are considered, then the length of execution of a logic program increases well beyond acceptable notions of computability. Therefore in SLP function symbols cannot be used, thus syntax is the same as the syntax of DATALOG. Note that in this case the Herbrand universe is finite, the set of ground atoms (Herbrand base, as it is called in the logic programming community) is finite. Therefore the first order logic of that language can be transformed into propositional logic.

The stable models of a logic program can be understood as follows. Let P be a logic program, and let be P_g be the set of the ground instances of clauses in P. For the sake of simplicity we identify models with their positive diagrams, that is with the set of positive ground atoms. Let M be such a set of positive ground atoms, it is called a stable model of P if M coincides with the least model of the reduct P_g^M of P_g with respect to M. It is obtained from P_g in two steps:

- for every ground atom $p \in M$ all the clauses containing $\mathbf{not}(p)$ are omitted,
- from the body of every remaining clause the negative literals are omitted.

Clearly M is a model of P too. Stable models have several "nice" properties, which we cannot explain here - these are defined and discussed in the literature about different theories about negation in logic programming. However, we mention that the definition of stable models - as the other constructions in this field - is greatly inspired by the developments in the field non-monotone logics, which we discuss later.[118], which is an easy and interesting introduction into stable logic programming, recommended to the interested reader. Another related reference is [19].

SLP is a basically different paradigm from the traditional Horn logic programming from the point of view of programmers, too. While traditionally the result of a computation is a term (or some terms), in the answer set programming the result are sets of atomic statements (facts). An example: traditionally

a path in a graph is expressed as a list of edges, while in answer set programming as a set of edges. In the traditional case the result holds in every intended model (the term belongs to every fixed point of the definition), in answer set programming different results are supplied for different intended models.

There are software implementations to compute answer sets, e.g.
DeReS (http://www.cs.engr.uky.edu/ai/ders.html),
dlv (http://www.dbai.tuwien.ac.at/proj/dlv),
smodels (http://www.tcs.hut.fi/Software/smodels/).

With the development of the field of knowledge based systems new types of reasoning, different from deduction, were applied, like abduction and induction. Soon the logic programming languages corresponding to them have appeared:

- *Abductive logic programming* is an expansion of the traditional one. Beside the logic program a set of atomic formulas is given, which can be taken as hypothesis. If the deduction fails, some of these atoms can be used as input clauses to continue the derivation. Then the result has to be understood "assuming hypotheses ...". About the application of abductive logic programming see e.g. [105].

  Inductive logic programming "is concerned with the development of inductive learning algorithms that use first order definite clause logic as the knowledge representation" - this is how the Introduction of [125] defines the field. Inductive logic programming reverses the task of logic programming: positive and negative examples are given, and a logic program has to be synthesized which computes all the positive examples and none of the negative ones. Background knowledge in the form of definite clauses can help induction. Inductive logic programming can be used in *knowledge extraction* (data mining) as well as in *optimizing logic programs*. For a detailed discussion see [124].

A logic programming language is a proof procedure, therefore the PE definitions as programs and the selected calculus have to be completed by a *search strategy*. The problem of search strategy is presented here for definite clause programs. Because of the restricted set of formulas used in the proofs the search space turns to be of a tree structure, more precisely an *and-or tree*. There are two questions to be decided by a search strategy:

1. If a goal is given in the form of $A_1 \wedge \ldots \wedge A_n \rightarrow$, it has to be decided which literal A_i has to be eliminated first (which has to be resolved upon). The node in the search tree corresponding to this decision is an **and** node, since all the literals has to be eliminated. In the traditional definition of PROLOG the goals - as well as all the clauses - are represented as sequences of literals, and the *first literal* is selected.

2. For a selected literal a definite clause has to be selected, which can be resolved upon the literal. This selection is represented by an **or** node.

In the traditional definition of PROLOG the **and** nodes do not occur in the search tree, since the selection function determines the literal to be eliminated. So, an *or tree* is obtained. The definite clauses with the same relation symbol

in their heads[10] are ordered, and they are selected in this order. The search strategy is *depth first* traversing of the search tree. Notice that this search may lead the execution into an infinite branch even if the empty clause can be generated, spoiling completeness. However, the programmers can influence the process of the search by the order of clauses and the order of literals in the clauses - therefore it is the programmer's responsibility to write programs which do not loop[11]

The fact that the logic program interpreters (compilers) are based on search suggests the application of *parallel* execution. In that case the and-or tree representation is adequate. Two kinds of parallelism are distinguished:

- if branches coming from an **and** node are traversed parallel, we speak about *and-parallelism*;
- if branches coming from an **or** node are traversed parallel, we speak about *or-parallelism*.

While the or-parallelism does not raise theoretical problems, and-parallelism does so. The difficulty is that literals in a goal may contain the same variable. The two occurrences have to bind the same way, so the two branches cannot be traversed independently of each other.

An interesting version of logic programing is the so called *Tabled Logic Programming*, or shortly *tabling*. At a very high level, during the computation of a goal, each sub goal S is registered in a table the first time it is called, and unique answers to S are added to the table as they are derived. When subsequent calls are made to S, the evaluation ensures that answers to S are read from the table rather than being re-derived using program clauses. Even from this simple description, a first advantage of tabling can be seen, namely, that it provides termination to various classes of programs. One powerful feature of tabling is its ability to maintain other global elements of a computation in the "table," such as information about whether one sub goal depends on another, and whether the dependency is through negation. By maintaining this global information, tabling can be used to evaluate normal logic programs under the Well-Founded Semantics. To learn more about tabling, visit the home page of the XSB Research Group: `http://xsb.sourceforge.net`, or `http://www.cs.sunysb.edu/~warren/xsbbook/book.html`, the latter being the draft of a book about programming in Tabled PROLOG.

"Pure"[12] logic programming is applicable to computation on abstract data types constituting a Herbrand universe. However, abstract data types and data types, involving real computation (e.g. numbers) are mostly used together. Even in the first variants of PROLOG there were so called "built in" predicates evaluating arithmetic expressions. However, this evaluation can be performed only if all the variables in the expression are bound. *Constraint logic programming* extends this feature with constraints satisfaction. The constraints (mainly in the form of equalities, inequalities) encountered in the process of proving the goal are placed into a set called *constraint store*. If a variable occuring in a

[10] They form a "partition" in the *PROLOG* terminology.
[11] Moreover commands to effect search are included in PROLOG.
[12] A logic program is *pure* if every step of the execution is executed by an inference rule.

constraint gets bound, the actual value is substituted in the constraint, and it is evaluated, if possible. If the evaluation gives false, the execution backtracks. When there is no literal to be eliminated by the inference rules, the system of the collected equations and inequalities is solved by mathematical methods. An operational semantics is given to constraint logic programming. If the reader wants to learn more about to constraint logic programming, [138] is suggested.

There are several constraint logic programming languages, like ECLiPSe (http://eclipse.org/), GNU PROLOG (http://www.gprolog.org/), Oz (http://www.mozart-oz.org/), SWI-Prolog (http://www.swi-prolog.org/). GNU Prolog and SWI-Prolog are free software.

Chapter 6

KNOWLEDGE BASED SYSTEMS

One of the main fields of Artificial Intelligence (AI) is automatic problem solving. In the last decades this field has given birth to several practical results, first *expert systems*, later *knowledge based systems* became a solid part of software industry.

A knowledge based system basically consists of two constituents: the *knowledge base* and the *reasoning engine*. In the case of the traditional program systems the domain specific knowledge is hidden in the code; therefore it is difficult to verify it, and it is almost impossible to maintain the system. On the other hand, in the case of knowledge based systems the knowledge is stored in the knowledge base in a descriptive form. The content of the knowledge base is interpreted by the reasoning engine. [134] provides a thorough and practical study of logic based problem solving.

It stands to reason to chose a logic as a *knowledge representation language* - in this case the reasoning engine is a corresponding theorem prover for the selected logic. However, there are difficult problems, the most important ones being listed below.

- The *combinatorial explosion* may spoil the effectivity of the theorem prover as a reasoning engine. Therefore sometimes specific segments of logics are considered as a knowledge representation language, like logic programming languages, and description logics (later will be discussed).

- The amount of the represented knowledge is usually big, therefore, search for the relevant knowledge related to a problem or a query is one of the greatest problems. Therefore soon it was realized that knowledge must be structured[1]. First different associative knowledge representation methods were worked out, like associative networks and frames. Recently these tools are called structured inheritance networks. The most successful version was the KL-One knowledge representation language. For its theoretical foundation a logic called *Description Logic* (DL) was worked out - however, soon it was discovered that DL itself is an adequate knowledge representation language. DL is detailed later.

[1] Notice that we have met similar situation when software specification was discussed.

There is an approach in logic called *labelled deduction*, which introduces and studies structures of formulas as "data base" instead of set of formulas as theories (see [72]).

- In most domains of the practice (like medicine, geology etc.) the rigidity of logics hinders its use as a knowledge representation language. There are no clear cut definitions, the statements are meant "in general" or "normally". However, tools for exception handling are missing from traditional logics. To compensate the lack of exception handling a new branch of proof systems was developed: *non-monotonic reasoning*, which is discussed in the next section.

- Again, in most domains the characteristic reasoning of the discipline is not deduction as formalized in mathematical logics. Therefore, different ways of *plausible reasoning* - borrowed from the field of philosophical logic - are studied by AI researchers and are used in knowledge based systems. The methodical development of knowledge based systems may also use some kind of logics, e.g. logic based induction may help knowledge acquisition.

6.1 Non-monotonic reasoning

6.1.1 The problem

To illustrate the need of non-monotonic reasoning let us see the "paradigmatic" example that is the sentence "Birds fly.", which is translated into logic with the use of a bounded universal quantifier as $\forall x(bird(x) \rightarrow fly(x))$. However, what happens if the discourse is about an ostrich? To overcome the problem, inference rule $\forall x \alpha(x) \vdash \alpha(x/t)$ is replaced with inference rules that do not allow substitution without constraints.

Let us look at some theories.

Historically, the first one was Reiter's *default logic* (defined in [139]), in which new rules can be formed as $\langle\langle \alpha, \beta \rangle \beta \rangle$ with the meaning "*if α is proved and β is consistent with the set of sentences generated so far, conclude that β is true*".

Another approach is *circumscription* (defined in [119]), which collects exceptions (maybe of different types) and an inference step can be executed only if exceptional situation are excluded. If we go back to the paradigmatic example, the sentence about birds may be formalized as $\forall x(bird(x) \wedge \neg abnormal_b(x) \rightarrow fly(x))$, and sentences like $\forall x(ostrich(x) \rightarrow abnormal_b(x))$ have to be added. The extensions of the different *abnormal* predicates must be the minimal defined relations - notice the similarity with logic programming. It is not suitable to use equivalence in the definition of abnormality, since the content of the knowledge base generally is not closed, later new exceptions have to be added. If equivalence was used, it would mean the modification of a sentence, however, using the implicative form, only a new sentence has to be added.

The approaches outlined above are basically ad hoc inference rules, and their practical use is doubtful: sometimes search strategy may effect what sentences are inferred.

However, the introduced rules spoil the monotonicity of the provability relation. Let us go back to the paradigmatic example, and let us suppose that Koko is a bird ($bird(Koko)$). If nothing else is known about this bird, we can conclude that Koko flies ($fly(Koko)$). If later we learn that Koko is an ostrich (so $abnormal_b(Koko)$ is true), the inference turns out to be false - the derived sentence (and all of its consequences) have to be deleted. Notice that non-monotonicity not only goes against the purity of proof systems as defined so far, but the revision of the knowledge base causes practical problems, too.

The problem of non-monotonic reasoning has been a challenge also for logicians, and new logics are worked out, like *autoepistemic logic*, outlined below. Moreover several logics rooted in different motivations can be considered non-monotonic ones, like *probability logic* or the *logic of conditionals*.

Before discussing some questions in a more detailed way, let us speak about the root of the problem. The basic fallacy is that sentences as "Birds fly" are not variants of universal sentences like "Every bird flies". Linguists call sentences like "Birds fly" *generic* sentences. Such sentences state a characteristic feature of a *gens*, a *species*, a *group*. Sentence "Birds fly" means that the ability of flying is a characteristic feature of birds, but it cannot be instantiated for every bird. It can be understood as "Normally birds fly", and such a reading is the usual in the context of non-monotonic logics.

6.1.2 Autoepistemic logic

Autoepistemic logic is concerned with beliefs and changes of beliefs. It is a non-monotonic logic, that is, the set of beliefs that are justified on the basis of given premises does not necessarily increase monotonically as the set of premises increases.

Databases *based on autoepistemic logic* have the ability to record whether their knowledge about a fixed object is complete or not. In Artificial Intelligence autoepistemic logic is intended to model the reasoning of an ideally rational agent. Let us see a simple example.

Let us assume that one of the basic principle of the agent's thinking is "*If I do not believe something, it does not holds*". So far she has not learnt that "*My first teacher, Mr. U died*", Since she has no basis to think so, she does not believe it. Therefore she thinks that "*Mr. U lives*". Later she learns that "*Mr. U died*". Then she has to update her knowledge and beliefs about the world:

1. "*Mr. U lives*" and all of its consequences has to be omitted.
2. "*Mr. U died*" has to be added.
3. "*I believe that Mr. U died*" has to be added.
4. The new set of knowledge and beliefs has to be closed under logical consequence.

Now, we sketch some basic concepts of autoepistemic logic. Our approach to autoepistemic logic is that of *proof theory*.

The language of this logic is like an ordinary modal (propositional) logic with the difference that the operator $\Diamond$ is missing and $\Box$ is denoted by B. Definition of formulas is the same as in modal logic.

The meaning of the unary operator B is "believed", so the meaning of the formula $B(P)$ is "P is believed", where P is a propositional symbol.

A central concept is the concept of *theory* (or *stable* theory).

6.1 Definition *A set T of formulas is said to be a stable theory in L if the following three conditions hold:*

(i) *if $P_1, \dots P_n \in T$ and $P_1, \dots P_n \vdash Q$ then $Q \in T$ (where $\vdash$ means ordinary logical consequence),*

(ii) *if $P \in T$ then $B(P) \in T$,*

(iii) *if $P \notin T$ then $\neg B(P) \in T$.*

The concept of stable theory corresponds to theory in ordinary logic. Condition *(iii)* is remarkable. It is an inference rule of *non-standard* form: if P is *not* a theorem then Q *is* a theorem where $Q = \neg B(P)$ (a usual rule of inference is of the form: if P *is* a theorem then Q is also a theorem).

An autoepistemic theory represents the information we have and our belief in a given stage (moment). It is *maximal* in the sense that no further conclusions could be drawn in this stage.

Even if we have a new piece of information, extending the actual stable theory with it is impossible. In this case another stable theory is needed. Theories can *change essentially stage by stage*, when certain formulas are removed and others are added. As in classical logic, different axioms (premises) can generate different theories, the same conclusion could be true in one theory and false in another theory.

The following theorem concerns a stable theory *generated* by a set Σ of premises.

6.2 Theorem (stable extension) *An autoepistemic theory T is a stable extension of the formula set Σ if it is the least set of sentences satisfying the following conditions:*

(i) $\Sigma \subseteq T$,

(ii) *T is closed under ordinary logical consequence ($\vdash$ relation of classical logic),*

(iii) *if $P \in T$ then $B(P) \in T$,*

(iv) *if $P \notin T$ then $\neg B(T) \in T$.*

Clearly, the stable extension of Σ is the *minimal* stable theory containing Σ.

The concepts of stableness can be characterized using *semantics*, too. It can be proven that a somewhat modified semantics of the modal logic $S5$ can be considered as a possible semantics of autoepistemic logic.

6.1.3 Non-monotonic consequence relations

While in the study of non-monotonic logics first different calculi were suggested, the research programme of focusing on the *consequence relation* of non-monotonic logics was first proposed by [73]. This proposal was elaborated by D. Lehman and his co-workers by creating a theory of non-monotic consequence relation comprising both proof and model theoretical aspects (see e.g. [108]).

Let us introduce a symbol $\vdash_n$ as a binary relation between formulas, and let us call expressions like $\alpha \vdash_n \beta$ *conditional assertions*, with the following intuitive meaning: "*if α holds, then normally β holds*". Notice that in a conditional assertion α has to be true according to the semantics of classical logic, and the word "*normally*" is applied only to β. Using our example: "Birds fly" can be written as $bird(x) \vdash_n fly(x)$, or $b \vdash_n f$ in propositional form. The symbol $\vdash_n$ is a metapredicate, it does not belong to the alphabet of the object language. However, we stipulate that conditional assertions occur in the knowledge base. A non-monotonic inference rule infer conditional assertions from conditional assertions. Notice also that this interpretation of non-monotonic reasoning is in accordance with the interpretation of generic sentences in natural languages. Also, some kind of concept of a "*normal world*" can be defined:

a model $\mathcal{A}$ is normal w.r.t. α if and only if for every β which is the consequence of α ($\alpha \vdash_n \beta$), $\mathcal{A} \models \beta$ holds.

[73] proposed to accept three characteristic features of the non-monotonic consequence relation (compare them with the properties of the provability relation):

$$\alpha \vdash_n \alpha \qquad \text{Reflexivity}$$

$$\frac{\alpha \wedge \beta \vdash_n \gamma \qquad \alpha \vdash_n \beta}{\alpha \vdash_n \gamma} \qquad \text{Cut}$$

$$\frac{\alpha \vdash_n \beta \qquad \alpha \vdash_n \gamma}{\alpha \wedge \beta \vdash_n \gamma} \qquad \text{Cautious Monotonicity}$$

The above presented features characterize the non-monotonic consequence relation independently of the logical connectives of the logic - that is why the rule of the substitution of equivalent formulas is missing. [108] completes the characteristics of non-monotonic consequence relation with the following two rules:

$$\frac{\vdash \alpha \to \beta \qquad \gamma \vdash_n \alpha}{\gamma \vdash_n \beta} \qquad \text{Right Weakening}$$

$$\frac{\vdash \alpha \leftrightarrow \beta \qquad \alpha \vdash_n \gamma}{\beta \vdash_n \gamma} \qquad \text{Left Logical Equivalence}$$

Notice that Left Logical Equivalence is weaker than Left Weakening which holds with respects to the traditional provability relation: $\frac{\vdash \beta \to \alpha,\ \alpha \vdash \gamma}{\beta \vdash \gamma}$. Substituting Left Logical Equivalence for the classical rule of Left Weakening is important in characterizing the non-monotonic consequence relation, as it can be illustrated by our paradigmatic example.

Let us assume that we have the following axioms: $\forall x(ostrich(x) \to bird(x))$, $\forall x(madár(x) \leftrightarrow bird(x))$ and conditional assertion $bird(x) \vdash_n fly(x)$ is in the knowledge base. It can be inferred by Left Logical Equivalence that $madár(x) \vdash_n fly(x)$, however $ostrich(x) \vdash_n fly(x)$ can not.

6.3 Definition (cumulative consequence relation) *A consequence relation is said to be* cumulative *if and only if it contains all instances of the Reflexivity axiom and is closed under the inference rules Cut, Cautious Monotonicity, Right Weakening and Left Logical Equivalence.*

Cumulative consequence relations can be characterized by semantic means too, namely by a generalization of Shoham [1984]. The intuitive idea is that there is a "more preferred" relation on the class of models, and a conditional assertion $\alpha \vdash_n \beta$ holds if and only if β holds in the most preferred models of α. [108] generalizes this idea. The concept of state is introduced, which is a set of models, and the "more preferred" relation are defined between states.

6.4 Definition (cumulative model) *A triple $\langle S,\ l,\ \prec\rangle$ is a cumulative model if and only if*

- *S is a set called the set of states,*
- *l is a function labeling every state with a non empty set of models,*
- *$\prec$ is a binary relation on S satisfying the smoothness condition defined below.*

For the next definition the notion *minimal* is generalized for arbitrary an binary relation $\prec$: s ($s \in A$) is minimal w.r.t. $\prec$ if and only if there is no $z \in A$ such that $z \prec s$.

6.5 Definition (smoothness) *Let A be a set, and let $\prec$ be a binary relation on A. It is said that $B \subseteq A$ is* smooth *w.r.t. $\prec$, if and only if for every element $t \in B$ if t is not minimal w.r.t. $\prec$, then there is an element s of B such that s is minimal w.r.t. $\prec$.*

A state s satisfies a formula α if and only if every model in $l(s)$ satisfies α. In the following definitions $\widehat{\alpha}$ denotes the set of states satisfying α.

6.6 Definition (smoothness condition) *A relation $\prec$ in a cumulative model $\langle S, l, \prec\rangle$ satisfies the smoothness condition if and only if for every formula α the set $\widehat{\alpha}$ is smooth w.r.t. $\prec$.*

6.7 Definition (cumulative consequence relation)
Let a cumulative model $W = \langle S, l, \prec\rangle$ is given. The consequence relation generated by W (denoted by $\Vdash_n^W$) is as follows:
$\alpha \Vdash_n^W \beta$ if and only if for any s minimal in $\widehat{\alpha}$ s satisfies β.

6.8 THEOREM (representation) *A consequence relation is a cumulative consequence relation if and only if it is generated by some cumulative model.*

The concept of a cumulative consequence relation can be refined adding new plausible rules; similarly, the concept of a cumulative model can be modified by restricting the function l and/or the relation $\prec$.

It is interesting that for any proposed non-monotonic calculus there can be found some plausible property of the consequence relation which is not satisfied. [108] quotes some properties which are not satisfied even their proposed system. These are different from the inference rules presented so far, namely they concern the *absence* of certain pairs from the consequence relation. Let us see an example:

$$\frac{\alpha \not\vdash_n \gamma \qquad \beta \not\vdash_n \gamma}{\alpha \vee \beta \not\vdash_n \gamma}$$ Disjunctive Rationality

We may conclude that the research of the consequence relation provides a deeper insight in non-monotonic logics; however, we do not have yet a system which would satisfy all the plausible requirements on non-monotonic reasoning.

References to Section 6.1 are, for example: [69], [108].

6.2 Plausible inference

Logic was developed with the purpose of characterizing sound human reasoning in general. However in the 20. century research in logics provided the formalization of special reasoning characteristic of different disciplines, domains. Moreover, the formal reasoning can be turned into practical tools in knowledge based systems. Therefore several ways of *plausible reasoning* got into the focus of research, namely induction, abduction, retroduction, different kinds of similarity based reasoning (analogical reasoning, case based reasoning). It is hard to clarify even the concept of plausible reasoning. Strictly speaking, every non sound reasoning belongs to it. Plausible reasoning is sometimes characterized as a kind of reasoning in which the result of some plausible proof can be defeated (refuted). If this property is in focus, the term *defeasible* reasoning is used. In D. M. Gabbay's opinion the main characteristic of plausible reasoning is that different chains of inference for and against a statement have to be weighed to reach a decision.

In another approach plausible reasoning is rather generating hypotheses than inferring a statement. So the scheme of plausible reasoning is as follows:

1. generating hypotheses,

2. accepting the most plausible one - if human intuition does not help, some kind of search strategy can do it,

3. testing the accepted one - this can be done by deduction.

For the first step, for hypotheses generation special logical calculi may be introduced. We only discuss *induction* and *retroduction* (or abduction[2]) in short.

Induction is a well known term of philosophical logic, it means to extract general rule from examples. In AI induction can be used in several fields such as machine learning, knowledge extraction.

The term retroduction as a way of reasoning, was first used by the outstanding American mathematician, Peirce [131]. It is a backward reasoning searching for the cause of some phenomenon. Such kind of reasoning is used in diagnostic problems (in medicine as well as in technical problems) and crime detection too.

Both induction and retroduction can be considered to be the inverse of deduction: the inference rules have to be reversed. Such a calculus keeps falsity instead of truth. To show an example of an non deductive inference rule, and to illustrate the difference between the ways of reasoning, let us see the case of Modus Ponens[3]:

$\alpha \to \beta,\ \alpha \vdash \beta$	deduction
$\alpha,\ \beta\ \vdash_{ind} \alpha \to \beta$	induction: if in the cases when α holds, β also holds, then $\alpha \to \beta$ can be assumed as a hypothesis
$\alpha \to \beta,\ \beta\ \vdash_{ret} \alpha$	retroduction: knowing the rule $\alpha \to \beta$ and having β, α can be assumed as a hypothesis

Plausible inferences are greatly different from proofs in logics discussed so far. Clearly the main difference is that the inferred statement is only a plausible consequence, even in the case when the axioms used in the inference hold. So we cannot speak about correctness and completeness of a calculus. However there are other differences too.

Note that the above rules for induction and retroduction outline only the logical nature of the plausible inference in question. The inference rules to be applied in a knowledge based system require a lot of conditions. Let us consider induction. From only one pair of formulae α, β we cannot infer $\alpha \to \beta$. The number of such pairs used in the inference characterizes the *strength* of the inference. Note that such notion as the "strength" of a proof is meaningless in a logic with a correct calculus. Another difference is that the number of inferences of a statement is an important factor in accepting the statement. These features can be considered as elements of the search strategy in a plausible inference method.

Another important factor is that plausible reasoning can be used only if it is based on an ontological basis. In the case of induction, from formulae α, β we can infer $\alpha \to \beta$ as well as $\beta \to \alpha$. There must be some factor outside of logic that allows us to select one of them. Let us see an example. The fact that in the stomach of patients having gastric ulcer there are special bacteria was the base of the hypothesis that this kind of bacterium causes gastric ulcer. Purely on a logical base the following hypothesis may stand as well: this kind of bacterium can survive in a stomach only if there is ulcer there. Clearly, the reason, why the first conclusion was inferred is that there is a bias in the field of medical research

[2]The word abduction is used in different meanings in AI literature - that is why we use the term retroduction, which is introduced parallelly with abduction by C. S. Pierce, see eg. [131]

[3]Note that non-monolotic reasoning is realy a way of plausible deduction - this was discussed in a seperate section only because of historical reasons.

that diseases are usually caused by pathogenes. Such bias can be expressed by a formula. However, in such cases the rules of plausible inference have to be used together with other kinds of inference rules. So far a proof system has been considered to consist of a pair of set of axioms and a set of rules, but in the case of plausible logics the situation is more complex. There are theoretically true sentences and empirical facts in the set of axioms, and there are correct and plausible inference rules as well. Moreover, inferred formulae have to be distinguished according to how they are inferred (how strong the inference is), and how many arguments provide some truth values to them. Therefore it is useful to introduce truth values of different kinds.

The classical truth values (true, false) are usually used for theoretical knowledge, empirical knowledge may has specific truth values. The empirical (factual) truth values are characterized by their degree of certainty, like "empirically true", "empirically false", "empirically contradictory", "uncertain". They can be precisely defined depending on the presence or absence of justifying and rejecting examples, e.g., by the following rules:

- a statement takes the truth value "empirically true", if there are arguments for it, and there are no arguments against it,
- a statement takes the truth value "empirically false", if there are no arguments for it, and there are arguments against it,
- a statement takes the truth value "empirically contradictory", if there are arguments for it, and there are arguments against it,
- a statement takes the truth value "uncertain", if there are no arguments for it, and there are no arguments against it.

The degree of certainty of an empirical statement can be characterized more precisely by some numerical scale. [16] writes about (and uses) several such values. Let p be the number of arguments "for", and c be the number of arguments "against". Then the fraction

$$\frac{p}{p+c}$$

expresses the "promixity" of the empirical truth. The scale is from 0 to 1, where 0 stands for "empirically false", and 1 corresponds to truth value "empirically true". A special case is uncertainty, when the value is $\frac{0}{0}$.

Another kind of truth value is computed as

$$\frac{p-c}{p+c}$$

It characterizes the nature of uncertainty. The scale is from -1 to 1, where -1 means "empirically false", 1 means "empirically true". All the other values expresses some kind of contradiction, 0 characterizing the pure case of "empirically contradictory", when there are precisely the same amount of arguments "for" and "against".

Since cognitive systems are in the focus of AI research, plausible reasoning methods are widely studied and used in several fields. In several cases non logical tools are used for plausible reasoning. However, there are also logical

proof systems suggested and used in practice. As an example, [16] describes a well developed general frame of logical systems for learning. Their general logic is the abstraction of JSM[4] method ([61], [62], a detailed description can be found in [16] too). Different versions of the method were used for data analysis in medical diagnosis, pharmacology and sociology. JSM method is really a hybrid method. It has logical rules, but the objects that the reasoning is about are represented by data structures (sets, graphs). Above these data structures a metric of similarity is defined, and inference depends on the measures of similarities.

References to Section 6.2 are, for example: [16], [149].

6.3 Description Logic

As we have wrote at the beginning of this chapter, for long different kinds of associative structures have been used for knowledge representation, now called *structured inheritance networks*. There are some logics to formulize such constructions, the most known and used is the *Description Logic* (DL).

The most important features of structured inheritance networks are as follows:

- The syntactic building blocks are concepts, individuals and the so called roles, expressing relations between individuals.
- Concepts are connected together in a network using the roles and metarelations (in particular subsumption relation between concepts).
- The expressive power of the language is restricted, that is, a rather small set of (epistemologically adequate) constructors are used for building complex concepts and roles.
- Implicit knowledge about concepts and individuals can be inferred automatically with the help of inference procedures. In particular, inference of the *subsumption* relationships between concepts and the *instance of* relationships between individuals and concepts plays an important role. The inference is controlled by the network mentioned in the second point.

The fourth feature clearly suggests to construct a kind of logic for structured inheritance networks. The third one shows that it will be a sublanguage of first order one. The first one characterizes the set of the non logical symbols of the language: they can be unary predicates (concepts), constants (individuals) and binary predicates (roles). The real problem lies in satisfying the second feature - this has been done by Description Logic.

The language of DL really consists of two sublanguages - one for describing general statements (formulae with free variables in classical first order logic), and one for describing facts about individuals. Therefore a DL theory consists of two sets of formulae, the first one is called Tbox (terminological box), the

[4] the letters J, S, M are the initials of John Stuart Mill, famous British philosopher. The method is founded on his rules for inductive reasoning.

second one is called Abox (assertion box). The language for Abox is a restricted 0th order logic, the real interesting topic is the language for Tbox. Below we discuss this language.

There is a great family of Description Logics, their common feature is that *predicates are used as constant symbols*, so no variables are used. The difference between the different DL languages is in their expressive power, that is, what kind of constructors are used to construct complex formulae. The following definition provides the most often used ones.

6.9 DEFINITION (constructors of DL languages) *Let C, D be concepts (unary relations), R be a binary relation of a DL language, $\boldsymbol{I}$ a first order model with universe $\boldsymbol{A}$, and let C^A, D^A, R^A denote the truth sets of C, D and R respectively. Let ∇ denote any of the following relation symbols: $\leq$, $\geq$, $=$. The following table defines the most widely used constructors to create complex DL formulae.*

name	formula	truth set
Top	$\top$	$\boldsymbol{A}$
Bottom	$\bot$	$\emptyset$
Intersection	$C \sqcap D$	$C^A \cap D^A$
Union	$C \sqcup D$	$C^A \cup D^A$
Negation	$\neg C$	$\boldsymbol{A} \setminus C^A$
Value restriction	$\forall R.C$	$\{a \in \mathbf{A} \ : \ \forall b \left(\langle a,b\rangle \in R^A \ \rightarrow b \in C^A\right)\}$
Existential quantification	$\exists R.C$	$\{a \in \mathbf{A} \ : \ \exists b \left(\langle a,b\rangle \in R^A \ \wedge b \in C^A\right)\}$
Numerical restriction	$\nabla n R.C$	$\left\lvert\{a \in \mathbf{A} : \exists b \left(\langle a,b\rangle \in R^A \wedge b \in C^A\right)\}\right\rvert \nabla n$

Note that the above defined formulae can be transformed into the first order logic. Let us see an example. If Φ denotes the transforming function, the transformed version of value restriction is $\forall y \left(R(x,y) \rightarrow \Phi\left(C\left(y\right)\right)\right)$. Every transformed formula can be written using of two variables, with the exception of numerical restrictions. This is very important, because the first order language with two variables is *decidable*.

So far the equivalents of open formulae are defined. The following definition define the sentences that can be used in DL, called Tbox axioms. Notations in the previous definition are used and S denotes a binary relation symbol.

6.10 DEFINITION (Tbox axioms) *The following axioms can occur in a Tbox.:*

name	*axiom*	*meaning*
concept inclusion	$C \sqsubseteq D$	$C^A \subseteq D^A$
concept equality	$C \equiv D$	$C^A = D^A$
role inclusion	$R \sqsubseteq S$	$R^A \subseteq S^A$
role equality	$R \equiv S$	$R^A = S^A$

The language of Abox is similar to the first order language, as the following definition shows. Let C be an arbitrary concept, and let R be a binary relation, a and b be constant symbols.

6.11 Definition (Abox formulae) *Formulae in the form* $C(a)$ *and* $R(a,b)$, $\neg R(a,b)$ *are Abox formulae.*

It can be seen that Abox formulae depends on the Tbox: the formula has to be created in the Tbox, and only then we can tell that the formula stands for a constant. Since Tbox formulae are equivalent to formulae of one free variable, only formulae about one constant can be formulated in the language defined so for. Therefore constructors to bridge the gap between Tbox and Abox are introduced, like the so called *fills* constructor R: a with the truth set $\left\{d \in \mathbf{A} : \langle d, a\rangle \in R^A\right\}$.

The version of DL defined so far - as the most widely used versions - is decidable. With respect to a particular set of Tbox axioms, proofs of the following properties are usually required:

- satisfiabilty,
- subsumption of concepts,
- equality of concepts,
- disjointness of concepts.

The usual Abox problems are the following ones:

- retrival problem, that is, "find a constant a such that $I \vDash C(a)$",
- realization problem, that is, "find the most specific concept C such that for a given constant a $I \vDash C(a)$".

Clearly all these problems can be reduced to unsatisfiability, so the semantic tableaux method can be used as a calculus tuned to DL logic. Theorem provers (like RacerPro, see [90], and Pellet, see [144]) work well for Tbox problems, but in the case of great Abox they fail. However, most practical realizations of DL involves huge Abox. Therefore using other calculi is an important research topic. A rich family of resolution based DL theorem prover methods has been worked out, see e.g., [115], [100].

Description Logic is used in several fields, however, its most important application is that the most widespread ontology description language OWL, is based on DL. A generally accepted definition of ontology is as follows:

An ontology is a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine-readable. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group.

Ontologies are used in several ways.

- A Tbox can be considered specification of a data base. Since a Tbox has a formal description language, its software realization can be used as a model of the data base. Interface can be built between the Tbox and the

data base, and inferences can be made about the data base. Moreover, an Abox can be used as data base. Such a system is called RDF[5] store, see

- Ontology can be considered a knowledge representation tool. Rules can be added to the DL description; usually, the ontology editors have also some rule description language.
- Ontologies are widely used in semantic web, [151] is a good introduction into this field.

References to Section 6.3 are, for example: [21], [151], [147].

[5]RDF is a simple knowledge representation tool, developed for web technology, see

Bibliography

[1] Abramsky, S., Gabbay, D.M., Mainbaum, T.S.E. (ed), Handbook of Logic in Computer Science I.: Background: Mathematical Structures, Oxford University Press, 1992

[2] Abramsky, S., Gabbay, D.M., Mainbaum, T.S.E. (ed), Handbook of Logic in Computer Science II.: Background: Computational Structures, Oxford University Press, 1992

[3] Abramsky, S., Gabbay, D.M., Mainbaum, T.S.E. (ed), Handbook of Logic in Computer Science III.: Semantic Structures, Oxford University Press, 1992

[4] Abramsky, S., Gabbay, D.M., Mainbaum, T.S.E. (ed), Handbook of Logic in Computer Science IV.: Semantic Modelling, Oxford University Press, 1992

[5] Abramsky, S., Gabbay, D.M., Mainbaum, T.S.E. (ed), Handbook of Logic in Computer Science V.: Theoretical Methods in Specification and Verification, Oxford University Press, 1992

[6] Abramsky, S., Gabbay, D.M., Mainbaum, T.S.E. (ed), Handbook of Logic in Computer Science VI.: Logical Methods in Computer Science, Oxford University Press, 1992

[7] Aczél P., "An introduction to inductive definitions" in: J. Barwise (ed.) Handbook of Mathematical Logic North-Holland, 1977, pp. 739-781

[8] Anderson J., D. Bothell, M. Byrne, C Lebiere, An Integrated Theory of the Mind, 2002

[9] Andréka, H., Ferenczi, M., Németi, I., Cylindric-like Algebras and Algebraic Logic, Springer, to appear

[10] Andréka H., Németi I., Completeness of Floyd Logic in: Bull. Section of Logic, 7, Wroclav, 1978, pp.115-120

[11] Andréka, H., Németi, I., General algebraic logic: A perspective on What is logic, in What is logical system, Oxford, Ed. D. Gabbay, 1994

[12] Andréka, H., Németi, I., Sain, I., Algebraic logic, Handbook of Philosophical Logic, Ed. Gabbay-Guenthner, vol II., 2002

[13] Andréka, H., Németi, I., Sain, I., Kurucz, A., Applying Algebraic Logic; A General Methodology. Lecture Notes of the Summer School "Algebraic Logic and the Methodology of Applying it", Budapest 1994, 67 pages

[14] Andréka, H., Németi, I., Sain, I., Universal Algebraic Logic, Springer, to appear

[15] Andrews, P. B., An Introduction to Mathematical Logic and Type theory, Academic Press, 1986

[16] Anshakov, O. M., Gergely T., Cognitive Reasoning, A Formal Approach, Springer, 2010

[17] Antonelli, A., "Non-monotonic Logic", The Stanford Encyclopedia of Philosophy (Summer 2003 Edition), Edward N. Zalta (ed.), http://plato.stanford.edu/archives/sum2003/entries/logic-nonmonotonic/

[18] Antoniou, G. and van Harmelen, F., Web Ontology Language: OWL, in 32, see: http://www.w3.org/TR/owl-features

[19] Apt, K., Bol, R., "Logic Programming and Negation: a Survey" in: Journal of Logic Programming 19-20, 1994, pp. 9-71

[20] Arora, S., Barak, B., Computational Complexity: A Modern Approach. Cambridge Univ. Press, 2009

[21] Baader, F., McGuiness, D.L., Nardi, D., Patel-Schneider, P.F. (eds), The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2003

[22] Beckert, B., R. Hanle, P. H. Schmit, Verification of object oriented software: The KeY approach Springer-Verlag Berlin, Heidelberg 2007

[23] Bell, J., Machover, M., A Course in Mathematical Logic, North Holland, 1977

[24] Ben, A. M., Amram, N.D., "Computational complexity via programming lannguages constant factors do matter" in: Acta Informatica, 37, 2, 2000

[25] Ben-Ari, M., Mathematical Logic for Computer Science (2nd ed.), Springer, 2001

[26] Bibel, W., Schnitt, P.H. Automated Deduction. A Basis for Applications, I., Kluwer, 1991

[27] Bjørner D., Hoare C.A.R., Langmaack H., "VDM and Z - Formal Methods": in: Software Development, Lecture Notes in Computer Science, 428, Springer Verlag, 1990

[28] Bjørner D., Jones C.B. (ed), The Vienna Development Method: the Meta-Language, Lecture Notes in Computer Science 61, Springer Verlag, 1978

[29] Blackburn, P., de Rijke, M., Venema, Y., Modal Logic, Cambridge University Press, 2001

[30] Blok, W.J., Pigozzi, D., Algebraizable logics, Memoirs of American Math. Soc., 396, 1985

[31] Bonacina, M. P., On theorem proving for program checking: historical perspective and recent developments. In: Proceedings of the 12th international ACM SIGPLAN symposium on Principles and practice of declarative programming ACM New York, 2010

[32] Boyer, R.C., Moore, J.S., A Computational Logic. Handbook., Academic Press, 1988

[33] Bramer, M. A., Logic Programming with PROLOG, Springer, 2005

[34] Burke, E., Foxley, E., Logic and its Applications, Prentice Hall,1996

[35] Caffera, R., Salzer, G.(Ed.), Automated Deduction in Classical and Non-classical Logics, Lecture Notes in Comp. Sci., 1761, Springer, 2000

[36] Chandru, V., Hooker, J., Optimizations Methods for Logical Inferences, Wiley, 1999

[37] Chiswell, I., Hodges, W., Mathematical Logic, Oxford Univ. Press, 2007

[38] Cholmicki J., Saake G. (ed), Logics for Databases and Information Systems Kluwer Academic Publishers,1998

[39] Chuaqui, T., Truth, Possibility and Probability, North Holland, 1991

[40] Cocchiarella, N. B., Freund, M. A. , Modal Logic: An Introduction to its Syntax and Semantics, Oxford Univ. Press, 2000

[41] Coniglio, M., E., Carnielli, W., A., Transfers between logics and their applications, Studia Logica, 72, 3, 2002

[42] Date, C. J., Logic and Databases: the roots of relation theory, Trafford Publ., 2007

[43] Dawsing, R.D., Rayword, V.J., Walter, C.D., A First Course in Formal Logic and its Applications in Computer Science, Blackwell Publ., 1986

[44] Dekhtyar, A., Subrahmanian, V.S., Hybrid probabilistic programs, J. of Logic Programming, 43, 3, 2000

[45] Dekker, P., Pauly, M., Logic and game, Journal of Logic, Language and Information, 11, 3, 2002

[46] Denecker, M., Kakas, A.C., Abductive logic programming, in J. of Logic Programming, 44, 1-2, 2000

[47] Ditmarsh, H. van, W. van der Hoek, B. Kooi, Dynamic Epistemic Logic Spronger Publishing Company, 2007

[48] Dovler, A., E. Pontelli eds., A 25 year Perspective on Logic Programming, Springler, Lecture Notes in Computer Science Vol. 6125, 2010

[49] Ebbinghaus, H., Flum, J., Finite Model Theory, Springer, 1999

[50] Ehrig, H., Mahr, B., Fundamentals of Algebraic Specification, Springer, 1985.

[51] Enderton, H., A Mathematical Introduction to Logic, Academic Press, 2000

[52] Ferenczi, M., Existence of partial transposition means representability in cylindric algebras, Mathematical Logic Quarterly, 57, 1, 2011, pp. 87-94

[53] Ferenczi, M., Mathematical Logic, Műszaki Kiadó, (in Hungarian), 2002

[54] Ferenczi, M., Non-standard stochastics with a first order algebraization, Studia Logica, 95, 2009, pp. 345-354

[55] Ferenczi, M., On homomorphisms between relation algebras, Algebra Universalis, 27, 1990

[56] Ferenczi, M., On representability of neatly embeddable cylindric algebras, J. of Applied and Non-classical logics, 3, 2000

[57] Ferenczi, M., On the representability of neatly embeddable CA's by relativized set algebras, Algebra Universalis, 4, 63, 2010, pp. 331- 350

[58] Ferenczi, M. Probabilities and measurable functions on cylindric algebras, in Cylindric-like algebras and Algebraic Logic, Springer, to appear

[59] Ferenczi, M., Sági, G., On some developments in the representation theory of cylindrical-like algebras, Algebra Universalis, 55, 2-3, 2006, pp. 345-353

[60] Ferenczi, M. The polyadic generalization of the Boolean axiomatization of fields of sets, Transaction of American Math. Society, to appear

[61] Finn, V.,K., Plausible inferences and plausible reasoning. J. Sov. Math. 56, 1991, pp. 319-323

[62] Finn, V.,K., The synthesis of cognitive procedures and the problem of induction. Nauchno-Technicheskaya Informacia, Ser. 2(1-2) pp. 8-45, 1999 (in Russian)

[63] Floyd R.M., Assigning meaning to programs in J.T. Schwartz (ed.) Proc. Symp. in Applied Mathematics, AMS, Providence, RI, 1967, pp.19-32

[64] Flum, J., Grohe, M., Parameterized Complexity Theory. Springer, 2006

[65] Font, J., M., Jannano, R., Pigozzi, D., A survey of abstract algebraic logic, Studia Logica, 74, 1-2, 2003

[66] Gabbay D.M., Guenther F. (ed), Handbook of Philosophical Logic 2. ed. Volume 1-9, Kluwer Academic Publisher, 2001, 2002

[67] Gabbay D.M., Hogger C., Robinson J. (ed), Handbook of Logic in Artificial Intelligence and Logic Programming I: Logical foundations Oxford University Press, 1993

[68] Gabbay D.M., Hogger C., Robinson J. (ed), Handbook of Logic in Artificial Intelligence and Logic Programming II: Deduction methodologies Oxford University Press, 1993

[69] Gabbay D.M., Hogger C., Robinson J. (ed), Handbook of Logic in Artificial Intelligence and Logic Programming III: Nonmonotonic Reasoning and Uncertain Reasoning Oxford University Press, 1994

[70] Gabbay D.M., Hogger C., Robinson J. (ed), Handbook of Logic in Artificial Intelligence and Logic Programming IV: Epistemic and Temporal Reasoning Oxford University Press, 1995

[71] Gabbay D.M., Hogger C., Robinson J. (ed), Handbook of Logic in Artificial Intelligence and Logic Programming V: Logic Programming Oxford University Press, 1996

[72] Gabbay D.M., Labelled Deductive Systems, Oxford Logic Guides 33.Calderon Press, Oxford 1996

[73] Gabbay D.M., Theoretical foundations for non-monotonic reasoning in expert systems. in K.R. Apt (ed) Proceedings of NATO Advanced Study Institute on Logics and Models of Concurrent Systems Springer, Berlin 1985, pp. 439-457

[74] Gallier, J., Logic for Computer Science. Foundation of Automatic Theorem Proving, Harper and Row, 1986

[75] Galton, A., Logic for Information Technology, Wiley, 1990

[76] Gentzen, G., Untersuchungen über das logische Schliessen, Math. Zeitschrift 39, 1996

[77] Gergely T., Szőts M.,On the incompleteness of proving patial correctness in: Acta Cybernetica 4 1978, pp. 45-57

[78] Gergely T., Úry L., A unique logical framework for software development Artificial Intelligence News Intelligent System Laboratorium, Moscow, pp. 62-81 1993

[79] Gergely T., Úry L., First-Order Programming Theories EATCS Monographs on Theoretical Computer Science 24, Springer Verlag 1991.

[80] Gergely T., Úry L., Program behaviour specification through explicit time consideration. in S.H. Lavington (ed): Information Processing '80, North Holland 1980

[81] Girard, J.Y., Linear Logic in: Theoretical Computer Science 50, pp.1-102

[82] Girard, J. Y., Proof theory and logical complexity I., Bibliopolis Naples, 1987

[83] Goerdt, A., Davis–Putnam resolution versus unrestricted resolution, in Logic and Combinatorics, Baltzer, 1992

[84] Goldblatt, R., Lectures on the Hyperreals, Springer, 1998

[85] Goldreich, O., Computational complexity: a conceptual perspective, Cambridge Univ. Press, 2008

[86] Groenendijk, Stokhof, Dynamic Predicate Logic. Towards a compositional and non-representational discourse theory ms, Amsterdam, ITLI 1987

[87] Guarino N., Giaretta P., Ontologies and Knowledge Bases: Towards a terminological Clarification, in: N.J.I. Mars (ed.) Toward a Very Large Knowledge Bases IOS Press, Amsterdam, 1995

[88] Gu, J., Franco, j., Purdom, P.W., Wah, B.W., Algoritms for the Satisfiability Problem, Cambridge Univ. Press, 2001

[89] Gurevich, Y., Semantics of program languages and model theory, in Algebra, Logic and Applications, 5, Langborne, 1993

[90] Haarslev, V., Moller, R., van der Straeten, R., Wessel, M., Extended Query Facilities for Racer and an Application to Software-Engineering Problems. In: Proceedings of the 2004 International Workshop on Description Logics (DL-2004), Whistler, BC, Canada, June 6-8. 2004, pp. 148–157

[91] Hacking, I., An Introduction to Probability and Inductive Logic, Cambridge University Press, 2001

[92] Halmos, P. R., Algebraic Logic, Chelsea Publ., 1962

[93] Harel, D., Tiuryn, J., Dynamic Logic, MIT Press, 2000

[94] Hayes I., VDM and Z, A Comparative Case Study in: Formal Aspects of Computing 4, 1992, pp. 76-99

[95] Henkin, L., Internal semantics and algebraic logic, in Leblanc, Truth, Syntax and Modality, Studies in Logic, 68, North Holland, 1973

[96] Henkin, L., Monk, J. D., Tarski, A., Cylindric Algebras I-II., North Holland, 1985

[97] Hirsch, R., Relation Algebras by Games, Elsevier, 2002

[98] Hoare C.A.R., An axiomatic basis for computer programming, in: Communication of ACM 12 1969, pp. 576-580

[99] Holzmann, G.J., The SPIN Model Checker : Primer and Reference Manual Addison Wesley, 2004

[100] Hustadt, U., Motik, B., Sattler, U., Reasoning for Description Logics around SHIQ in a resolution framework. Technical report, FZI, Karlsruhe, 2004

[101] Hu, T. H., Tang, C. Y., Lee, R.C.T., An avarage case analysis of a resolution principle in mechanical theorem proving, in: Logic and Combinatorics, Baltzer, 1992

[102] Jacky, J, The Way of Z: Practical Programming with Formal Methods Cambridge University Press, 1997

[103] Jones C.B., Systematic Software Development using VDM Prentice Hall, New York, 1994

[104] Joshi, R., Leino K.R., A semanticapproach to secure information flow in: Science of Computer Programming 37 2000, pp.113-138

[105] Kakas A.C., Riguzzi,F., Abductive Concept Learning in: New Generation Computing 18 2000, pp. 243-294

[106] Kanovei, V. G., Reeken, M., Non-standard Analysis, Axiomatically, Springer, 2004

[107] Knowledge Systems Laboratory, Stanford University: JTP: An Object-Oriented Modular Reasoning System; www.ksl.stanford.edu/software/JTP, July 24, 2003

[108] Kraus S., Lehman D., Magidor M., Nonmonotonic Reasoning, Preferential Models and Cumulative Logics Artificial Intelligence 44 1990, pp.167-207

[109] Kröger, F., S. Merz, Temporal logic and state systems Springer-Verlag Berlin, Heidelberg, 2008

[110] Kröger F., Temporal Logics of Programs, Spriger Verlag, Berlin, 1987

[111] Lamport, L., High-Level Specifications: Lessons from Industry. Springer Verlag, 2003

[112] Lamport, L., Paulson, L. C., "Should your specification language be typed?," SRC Research Report 147, Digital Systems Research Center, Palo Alto, CA, May 1997. Available at http://www.research.digital.com/SRC
http://citeseer.ist.psu.edu/article/lamport98should.html

[113] Lamport, L., Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers Pearson Education, Inc., 2002

[114] Libkin, L., Elements of Finite Model Theory, Springer, 2004

[115] Lukácsy, G. P. Szeredi. Efficient description logic reasoning in Prolog: the DLog system. Theory and Practice of Logic Programming. 09(03):,343-414, May, 2009. Cambridge University Press, UK.

[116] Maddux, R., Some varieties containing relation algebras, Trans. Amer.Math. Soc., 272, 1982

[117] Manzano, M., Extensions of First Order Logic, Cambridge Univ. Press, 1996

[118] Marek, V. and Truszczynski, M., Stable models and an alternative logic programming paradigm, in: The Logic Programming Paradigm: a 25-Year Perspective, 1999.

[119] McCharthy J., Circumscription: A form of non-monotonic reasoning in: Artificial Intelligence 13 1980, pp. 27-39

[120] Monk, J. D., Mathematical Logic, Springer, 1976

[121] Moore, R.C., Autoepistemic logic, in Non-standard Logics for Automated reasoning, Academic Press, 1988

[122] Morris J.M., Bunkeburg A., E3: A logic for reasoning equationally in the presence of partiality Science of Computer Programming 34, 1999

[123] Motik, B., Reasoning in Description Logics using Resolution and Deductive Databases. PhD thesis, Universitat Karlsruhe (TH), Karlsruhe, Germany (January 2006)

[124] Muggleton S., Foundation of Inductive Logic Programming Prentice Hall, Englewood Cliffs, N.Y., 1995

[125] Muggleton S., Page D. (ed), Special Issue on Inductive Logic Programming Journal of Logic Programming 40 N2/3, 1999

[126] Németi, I., Algebraization of quantifier logics, an introductory overview, Studia Logica 50, 3-4, 1991

[127] Németi, I., Some construction on cylindric algebra theory applied dynamic algebras of programs, Comp. Linguist., 14, 1980.

[128] Padawitz, P., Swinging types = functions + relations + transition systems Theoretical Cmputer Science 243 2000, pp. 93-165

[129] Papadimitriou, C. H., Computational Complexity, Addison-Wesley, 1995

[130] Pass, G., Probability logics, in Non-standard Logics for Automated Reasoning, Academic Press, 1988

[131] Peirce, C.S., Abduction and induction, in J. Bucher (ed) Philosophical Writings of Peirce, Dover, NY. 1995, pp.150-156

[132] Plotkin, L., Universal Algebra, Algebraic Logic, and Databases, Kluwer, 1984

[133] Pnueli, A., The temporal logic of programs. in Proc. 18th IEEE Annual Symp. on Foundation of Computer Science, Providence, RI 1977, pp. 46-57

[134] Poole, D., Mackworth, A., Goebel, R.: Computational Intelligence: A Logical Approach, Oxford University Press, 1998

[135] Prawitz, D., Natural Deduction Almquist & Wiksell, Uppsala, 1965

[136] Priest, G., An Introduction to Non-Classical Logic, Cambridge University Press, 2008

[137] Rasiowa, H., Logic, Algebra and Computer Science, Banach Center Publ., 1989

[138] Rossi F., P. van Beek, T. Walsh eds., Handbook of Constraint Programming, Elsevier, 2006

[139] Reiter R., Nonmonotonic reasoning. in Annual Reviews in Computer Science 2, Annual Reviews Inc. 1980, pp.147-186

[140] Robinson, J.A., Computational Logic, MIT Press, 1991

[141] Schonfield, W., Applications of relation algebras in Computer Science, Notices Amer. Math. Soc., 24, 1977

[142] Scott, D., Krauss, P., Assigning probabilitise to logical formulas, in Aspects of Inductive Logic, North Holland, 1966

[143] Serény, G., Gödel, Tarski, Church and the Liar, Bulletin of Symbolic Logic, vol.8, 2002

[144] Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y., Pellet: A practical OWL-DL reasoner. Web Semant. 5(2), 2007, pp. 51–53

[145] Smets, P., Mamdani, E.H., Dubois, D., Prade, H., Non-standard Logics for Automated Reasoning, Academic Press, 1988

[146] Spivey. J.M., Understanding Z: A Specification Language and its Formal Semantics. Cambridge Tracts in Theoretical Computer Science 3 Cambridge Universitz Press, 1988

[147] Staab, S., Studer, R., (eds.), Handbook on Ontologies.Springer 2009 (second ed.), Cambridge University Press, 2003

[148] Studer, R., Benjamins VR., Fensel D., Knowledge Engineering: Principles and methods IEEE Transactions on Data and Knowledge Engineering 25(1-2): 161-197, 1998

[149] Sun, R. ed., The Cambridge Handbook of Computational Cognitive Modeling, Cambridge Univ. Press, 2007

[150] Sun, R., The challenges of building computational cognitive architectures, in W. Duch, J. Mandziuk eds. Challenges in Computational Intelligence, Springer, 2007

[151] Szeredi P., G Lukácsy, T. Benkő. Theory and Practice of the Semantic Web. Cambridge University Press, 2011. (ready for publishing) (In Hungarian: A szemantikus világháló elmélete és gyakorlata, Typotex, 2005)

[152] Szőts M., A general paradigm of logic programming. in: Beyond Number Crunching, J. von Neumann Soc., Budapest, 1986

[153] Tarek Sayed Ahmed, Algebraic Logic, where does it stand today? Bulletin of Symbolic Logic. 11, 4, 2005, pp. 465-516

[154] Thayse, A. (ed.), From Modal Logic to Deductive Databases, Wiley, 1998

[155] Turini, F., Krzysztof, A.(Ed.) Meta-logics and Logic Programming, MIT Press, 1995

[156] Turner, P., Logics for Artificial Intelligence, Wiley, 1985

[157] Urquhart, A., Many-valued logics, In Handbook of Philosophical Logic III., Ed., D.Gabbay, Kluwer, 1986

[158] Urquhart, A., The complexity of propositional proofs, J. of Symbolic Logic, 1, 1995

[159] Urquhart, A., The relative complexity of resolution and cut-free Gentzen systems, in Logic and Combinatorics, Baltzer, 1992

[160] vanBenthem, J., Language in Action (Categories, Lambdas and Dynamic Logic), North Holland, Studies in Logic, 130, 1991

[161] Van Gelder A., Ross K., Schipf J., The Well-Founded Semantics for General Logic Programs in: Journal of ACM 38 1980, pp. 620-650

[162] Wand, M., A new incompleteness result for Hoare System in: Information and Control 56, 1978, pp. 72-99

[163] Wang, G., Zhou, H., Introduction to Mathematical Logic and Resolution Principle, ALPSC, 2010

[164] Zhou, N., H. Wang, X. Xu, P. Hung, J. Xie, Using TLA for modeling and analysis of Web services composition in: Ubi-Media Computing, 2008 First IEEE International Conference on 2008

Index