B4. Logical Perspective on Data and Knowledge

Lech Polkowski
Polish-Japanese Institute of Information Technology
Koszykowa 86, 02-008 Warsaw, Poland
and
Department of Mathematics and Information Sciences
Warsaw University of Technology
Pl. Politechniki 1, 00-661 Warsaw, Poland
e-mail: polkow@pjwstk.waw.pl

Andrzej Skowron
Institute of Mathematics
Warsaw University
Banacha 2, 02-097 Warsaw, Poland
e-mail: skowron@mimuw.edu.pl

Abstract. Logic understood as a study of mechanisms of inference, involving inference about knowledge from data, has evolved into many deductive schemes (languages) differing by understanding of semantics of inference $p \vdash q$. In this Section, basic schemes are outlined: classical calculi, many–valued logics, modal logics along with deductive mechanisms: axiomatized schemes, resolution in logic programming. An example of the system DATALOG is discussed. Also mentioned are various approaches to reasoning in inconsistent situations: non–monotonic logics, probabilistic logics, fuzzy, rough and mereological logics. In complex tasks of AI and KDD like pattern recognition or machine learning, inductive reasoning e.g. inductive logic programming is more frequent in applications aimed at defining relevant concepts and dependencies among them. We discuss basic aspects of reasoning with knowledge i.e. making inferences in a chosen logical language from a given knowledge base as well as basic aspects of reasoning about knowledge i.e. making also inferences about knowledge itself (e.g. concerning properties of knowledge like completeness, certainty or operating with knowledge like fusion).

Keywords: concepts, knowledge, logic, first order (predicate) logic, propositional logic, modal logic, deduction, induction, relations between data and knowledge, approximate reasoning.

B4.1. Data, Knowledge and their representation

Perception, description and analysis of real life phenomena have dominated intellectual activity of a human being; a fortiori, this activity has been assigned to machine systems exploiting various computing paradigms. Observation of a real life phenomenon may be passive or active: by the former we mean perceiving the phenomenon and possibly rendering its impression while by the latter we mean the process in which we create tools for a quantitative description of the phenomenon in terms of measurements, recordings, expert's knowledge etc. This

latter process leads to the record of the phenomenon in the form of data. Data may therefore be of many various types: numerical data, symbolic data, pattern data including time series data, etc. These types of data may be further made into more complex types e.g. arrays of numerical data or audio or video series (e.g. documentary films). The choice of a way of data collecting as well as a type of data depends on a particular problem which we are going to solve about the given phenomenon; this data elicitation process may have a great complexity and it is thoroughly studied by many authors [26]. Data elicited from a phenomenon should undergo a representation process in which they are modeled by a certain structure. This structure allows for efficient knowledge representation and reasoning about it in order to solve some queries or problems. The relationship of data and knowledge, in particular how knowledge can be acquired from data, has attracted attention of many philosophers and logicians cf. e.g. [61].

From logical point of view, data represent a model of a phenomenon i.e. a set of entities arranged in a certain space - time structure. Clearly, a real phenomenon may have associated with it many distinct data structures.

Usually, the nature of the phenomenon suggests us certain primitive concepts i.e. sets of entities in data structure in terms of which we build more complex concepts and we carry reasoning about the phenomenon. Logic provides a widely accepted representation of properties of data structures, concepts and their relationships or actions. A logic involves a set of formulas (well-formed expressions in a symbolic language) along with a family of relational structures (models) in which formulas are interpreted as concepts (i.e. sets of entities) of various relational complexities as well as a mechanism allowing us to reason about properties of models. The choice of the language of formulas may be critical: on one hand, this language should be expressive enough to render all essential concepts in data structures. On the other hand too expressive a language may cause too high complexity of inference process (the phenomenon of language bias in Machine Learning, Pattern Recognition, KDD [26], [44], [45], [49], [16]). The primitive data structures constructed according to a selected way(s) of recording a phenomenon present itself as possible models for various logics. Strategies for discovery of a particular logic, relevant for problems to be solved, present a challenge for KDD.

To illustrate the above ideas, we recall here a well–known logical language *DATALOG* used in relational databases (cf.[10], [69]).

Example: DATALOG

In *relational databases*, data are represented in two–dimensional tables called relations. Each row of the data table defines an instance of the relation among items occurring in this row. In DATALOG, relations are represented as *predicates*: for a relation R of arity n, we may introduce a *predicate symbol* P_R^n (often written down simply as R also) and we may define an *atom* (*an elementary formula*) $P_R^n(x_1, x_2, .., x_n)$. From atoms, more complex formulas may be constructed by means of *propositional connectives* : $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT) e.g. $P_R^n(x_1, x_2, .., x_n) \wedge P_S^m(x_1, x_2, .., x_m)$. In this way algebraic operations on

relations are rendered in DATALOG in symbolic, logical form. We may have other atoms, e.g. arithmetic, like $x + y > 1$ etc. In DATALOG, we may also define *rules* as implications of the form $P^n_R(x_1, x_2, .., x_n) \longleftarrow \wedge^k_{i=1} P^{n_i}_{S_i}(x^i_{i_1}, ..., x^i_{i_{k_i}})$; rules allow for symbolical rendering of a mechanism of new relation formation.

We have presented the syntax of DATALOG i.e. mechanisms of formula formation. The next aspect is its *semantics* i.e. meaning, interpretation of formulas. To interpret formulas, we must *evaluate* variables $x_1, x_2, ...$; to this end, we introduce a *valuation* v as a function which assigns to each variable x_i an element $a_i = v(x_i)$ in a specified set D called the *domain of interpretation* (for instance, D may be the set of objects occurring in data tables defining relations in our relational database). We have to select *truth values*, usually as T (True), F (False). Then we may define the meaning $[\alpha]_v$ of the atom α: $P^n_R(x_1, x_2, .., x_n)$ under the valuation v: $[\alpha]_v$=T in case $R(a_1, ..., a_n)$ holds i.e. $a_1, ..., a_n$ is a row in the data table representing R, and $[\alpha]_v$=F, otherwise. *Semantic rules* assign meanings to complex formulas e.g. $[\alpha \wedge \beta]_v$=T if and only if $[\alpha]_v$=T=$[\beta]_v$; $[\alpha \vee \beta]_v$=T if and only if $[\alpha]_v$=T or $[\beta]_v$=T; $[\neg\alpha]_v$=T if and only if $[\alpha]_v = F$.

Logical structure of DATALOG allows for procedures not allowed by the original structure of data tables by e.g. a possibility to define new concepts.

In this example the two facets of a logical system: syntax and semantics are clearly visible and the logical language clearly matches the original data structure.

In fitting a logic to a data structure, some important intermediate steps are to be taken:

- in the data structure, certain sets of entities (concepts) and relationships among them are selected giving admissible relational structures in data (cf. relation representation in DATALOG);
- mechanisms of inference about properties of these admissible structures are selected (e.g. some deductive systems (see below) like propositional language in DATALOG).

Inference mechanisms of a logic may provide us with descriptors of complex concepts implicit in data structures; for various reasons, the formulas of logic may not describe concepts in data structures exactly but approximatively only: one of reasons is that we may not know exactly the concept in question (we usually know some positive or negative examples of this concept - this is typical for Machine Learning, Data Mining and Knowledge Discovery). This makes it necessary to invoke in addition to deductive systems also inductive ones allowing us to build models of inference from sets of examples. This leads to new logical systems for approximate reasoning allowing us to carry out reasoning about properties of data structures on basis of uncertain, incomplete or insufficient data [*logics for reasoning under uncertainty*]. In these logics we encounter various phenomena not experienced by classical logics like non–monotonicity, necessity of belief revision etc. (see below). When a logic is selected which approximately specifies a data structure, this logic becomes an inference engine for using knowledge about a given phenomenon.

B4.2. Concepts

In general, the idea of a *concept* is associated with a set of entities; given a set U of entities, we call a concept any subset of the (universe set) U. For instance, in Example of Sect. 4.1, a concept may be any subset of the domain D of objects listed in data table. This notion of a concept is what may be called a *crisp* (*theoretical*) *concept*: the subset is understood here in the classical sense i.e. for each element of U we can decide whether it is in X or not. However, concepts in data structures are often *non-crisp* (*vague*): there are elements about which we cannot determine their membership in X with certainty. A typical example is provided by concepts expressed in natural language e.g. *high*: it may be a matter of dispute whether a man of height 175 cm is high or not. Also concepts known by examples only (observational concepts) are such. To cope with such concepts various theories have been proposed e.g. probabilistic and statistical reasoning [1], fuzzy set theory [73], rough set theory [52], multi-valued logics [58] etc.

Concept description may be twofold: *syntactical* as well as *semantical*. In classical case, syntactical description of a concept is provided by a formula of a logic. Semantical description relative to a model (the concept meaning) is provided by the meaning of this formula i.e. a set of entities in a model which satisfy this formula. This becomes more complex in cases for knowledge revision [12] e.g. for fuzzy or rough concepts where models learned from training data have to be revised after being tested against new cases.

A concept may also be characterized with respect to a set of formulas: given a crisp concept X and a set F of formulas, the *intension* of X with respect to F is the subset F' of F consisting of those formulas which are satisfied by each of elements of X; assigning to a subset F' of F the family of all elements which satisfy each formula in F', we obtain a set (concept) called the *extension* of F' (in particular, we may start with X and find the extension X' of the intension F' of X; here we work in the frame of Galois connections [71]). This idea becomes more complicated in case of non-crisp concepts where a formula is satisfied by an object in a degree usually less than 1.

In many applications there is need for analyzing dynamic structures involving changes of situations (concepts) by actions; from our point of view, actions are binary relations on concepts i.e. sets of pairs (pre-condition, post-condition) [62].

Reasoning about crisp concepts may be carried out by means of *classical deductive systems*. In non-classical cases, where observational concepts are only approximated by theoretical concepts, an important additional ingredient in reasoning is provided by some measures of similarity (distance) among the concept and its approximation as well as by some mechanisms for propagation of these closeness measures or uncertainty coefficients [74].

B4.3. Logic

B4.3.1 General view on logic

In spite of many views, often contradictory, on logic and its usefulness in Artificial Intelligence, in particular in Knowledge Discovery and Data Mining, it seems

that anyone should agree with the statement that logic gives us a mechanism for creating aggregates (collections) of statements in which one statement (called the *conclusion*) is the consequence of remaining statements (called *premises*) in the sense that whenever we believe the premises we should also believe the conclusion. The conclusion and the premises are then in the *consequence* relation. The process of passing from believed premises to the believed conclusion (*inference process*) is at heart of reasoning. Inference processes may be composed into chains of inferences and the overall inference mechanism may be very complex. Formal logic attempts at capturing essential features and properties of inferential mechanisms applied in many various contexts. Various logics differ with respect to the language in which they construct their statements, the way in which they construct their consequence relations, and the way in which they understand the notion of belief. For instance, in classical logics (like in DATALOG, 4.1), the belief is understood as the (absolute) truth therefore consequence relations in these logics lead from true premises to the true conclusion. On the contrary, in non-classical logics, belief may be understood as the probability of a statement to be true, the possibility of a statement to be true, the degree of belief in a statement to be true etc. therefore consequence relations express the degree of belief in the conclusion as a function of degrees of belief in premises.

In order to illustrate these aspects, we begin with a description of basic classical logical systems. The reader may use DATALOG as an example in the introductory part below.

B4.3.2 Syntax, semantics

Any logic needs a language in which its statements are constructed and its inference mechanisms are represented. Hence we should define an alphabet of symbols over which well-formed expressions (formulas) of logic are to be constructed. Usually, the process of constructing formulas is of a generative character: one starts with simple (elementary, atomic) formulas and applies some generative rules for producing more complex formulas. Syntactic characteristics of a logic involve a specification of an alphabet, a class of atomic formulas as well as rules for generating formulas. This purely syntactic aspect of logic has its counterpart in the semantic aspect dealing with the meaning of formulas and the semantic aspect of consequence relations. Building semantics for a logic involves therefore a certain world (or worlds) external to the set of formulas of the logic in which we interpret formulas assigning to each of them its meaning usually being relations in this world (worlds); a good example is a relational database providing a semantic frame for logic (e.g. DATALOG). With respect to this world (worlds), we can define truth values (in general, belief degrees) of formulas. When this is done, we can study semantically acceptable consequence relations as these inference mechanisms which lead from true premises to true conclusions (respectively, from premises in which we believe in satisfactory degree to the conclusions in which we believe sufficiently) (in both cases with respect to a chosen set of worlds).

With a logic we associate therefore two basic relations: the relation of syntactic consequence denoted $\vdash$, and the relation of semantic consequence (*entailment*) denoted $\models$.

Exemplary classical logical systems (calculi)

We review now some basic logical systems. We begin with propositional logic which deals with declarative statements. In this logic we may see in the simplest form all aspects discussed above.

Propositional logic

In this logic we consider *propositions* i.e. declarative statements like *London is the capital of Great Britain* or $2 + 2 = 3$ about which we can establish with certainty whether they are true or false. The calculus of propositions is effected by means of *propositional connectives* which allow for constructions of complex propositions from simpler ones. Formally, we begin with the *alphabet* consisting of a countably many propositional symbols $p_1, p_2, ..., p_k, ...$, functor symbols $\neg$,$\Rightarrow$ and auxiliary symbols: parentheses),(,],[. An *expression* is any word over this alphabet. The set of *formulas* of propositional logic is defined as the set X such that (i) X contains all propositional symbols (ii) X with all expressions u,v contains expressions $\neg u$ and $u \Rightarrow v$ (iii) if a set Y satisfies (i), (ii) then $X \subseteq Y$. To describe X, a generative approach is also used: one sets a set $A \subset X$ of formulas called *axioms* and one specifies *derivation rules* for generating formulas from axioms. Axioms may be chosen in many distinct ways; a simple axiomatics [42] consists of the following axiom schemes (meaning that in each of these expressions we may substitute for $p, q, r, ...$ any formula and we obtain an axiom formula):

(Ax1) $(p \Rightarrow (q \Rightarrow p))$;
(Ax2) $(p \Rightarrow (q \Rightarrow r)) \Rightarrow ((p \Rightarrow q) \Rightarrow (p \Rightarrow r))$;
(Ax3) $((\neg p \Rightarrow \neg q) \Rightarrow ((\neg p \Rightarrow q) \Rightarrow r))$.

The set of derivation rules consists of a single relation on expressions called *modus ponens* MP being the set of triples of the form $(p, p \Rightarrow q, q)$ meaning that: if $p, p \Rightarrow q$ are already derived from axioms, then q is regarded also as derived. The set of *theorems* is the set of formulas which can be obtained from instances of axioms by means of applying MP a finite number of times. The basic tool in efficient checking syntactic properties of propositional logic is the *Herbrand deduction theorem*: *For any set of formulas* Γ

$$\textbf{if } \Gamma, p \vdash q \textbf{ then } \Gamma \vdash p \Rightarrow q.$$

Now, we discuss semantics of propositional logic. We evaluate formulas with respect to their truth values: *truth* (denoted by T) and *falsity* (denoted by F) assuming that functors are truth-functional i.e. they are functions on truth values and they do not depend on particular type of a formula. Under these assumptions one can characterize functors semantically by means of tables. We give the

tables for $\neg, \Rightarrow$.

p	$\neg p$
0	**1**
1	**0**

Table 1. The negation functor $\neg$

p	q	$p \Rightarrow q$
0	**0**	**1**
0	**1**	**1**
1	**0**	**0**
1	**1**	**1**

Table 2. The implication functor $\Rightarrow$

Semantics of propositional logic is defined with respect to a *model* being the set of all boolean (i.e. 0, 1 - valued) functions (called valuations) on the set of propositional symbols: given a formula $\alpha(p_{i_1}, p_{i_2}, ..., p_{i_k})$ (which means that the propositional symbols $p_{i_1}, p_{i_2}, ..., p_{i_k}$ are the only variable symbols in α) and a valuation v we define the value $v(\alpha)$ with respect to α. An admissible state for α in the model is any valuation v such that $v(\alpha) = 1$. A formula α is *true* (is a *tautology*) when all states are admissible i.e. $v(\alpha) = 1$ for every valuation v.

Two important properties of this deductive system are: *soundness* (meaning that every theorem is true) and *completeness* (meaning that every true formula is a theorem). It is straightforward to check the soundness of propositional logic by induction on the formula length. Less obvious is the completeness of propositional logic established first by Gödel [42]. Propositional logic is *decidable*: for each formula it is sufficient to check finitely many partial valuations restricted to the finite set of propositional symbols occurring in this formula to decide whether the formula is true. It is also *effectively axiomatizable*: for each formula one can decide in a finite number of steps whether the formula is an instance of an axiom scheme. Soundness implies *consistency*: for no formula α both α and $\neg\alpha$ can be theorems.

Let us add finally that in practical usage additional functors are introduced, familiar from DATALOG : the conjunction functor $\wedge$ defined by taking $\alpha \wedge \beta$ as a shortcut for $\neg(\alpha \Rightarrow \neg\beta)$, the disjunction functor $\vee$ defined by taking $\alpha \vee \beta$ as the shortcut for $\neg\alpha \Rightarrow \beta$ and the logical equivalence functor $\leftrightarrow$ defined by taking $\alpha \leftrightarrow \beta$ as the shortcut for $(\alpha \Rightarrow \beta) \wedge (\beta \Leftarrow \alpha)$. Truth tables for these functors follow immediately from these definitions and tables 1,2.

Propositional reasoning turned out to be very effective for solutions of many KDD problems (see Boolean reasoning in Section B6) despite of the high computational complexity of the satisfiability problem of propositional calculus (it is NP-complete [22]).

Predicate logic

Propositional logic renders us good service by formalizing the calculus of propositions; however, in many practical situations, KDD applications including, we are concerned with properties of objects expressed as concepts i.e. sets of objects and with relations among these properties. In order to ensure the expressibility of relations e.g. inclusions (like *every ripe tomato is red*) we need to quantify statements involving object descriptors over concepts. The predicate logic is an extension of propositional logic enabling us to manipulate concept descriptors. We will write $P(x)$ to denote that the object denoted x has the property denoted P; the symbol P is a (unary) predicate symbol (cf. DATALOG, 4.1). With the expression $P(x)$ we associate two expressions: $\forall x P(x)$ and $\exists x P(x)$; the symbol $\forall x$ is called the universal quantifier and $\forall x P(x)$ is read *for each object x the property P holds* and the symbol $\exists x$ is called the existential quantifier and $\exists x P(x)$ is read *there exists an object x such that P holds for x*. Predicate calculus formalizes such utterances. For generality' sake, we will give a formal analysis of deductive systems known as first order theories of which predicate calculus is a specialization. To give a formal description of first order logical calculi, we begin with an alphabet which in general case consists of few types of symbols:

(i) *individual variables* $x_1, x_2, ..., x_k, ...$;
(ii) *individual constants* $c_1, c_2, ..., c_k, ...$;
(iii) *predicate symbols* $P_1^1, P_2^1, ..., P_{i_k}^k, ...$ where the upper index gives the arity of the predicate denoted thus;
(iv) *functional symbols* $f_1^1, f_2^1, ..., f_{i_k}^k, ...$;
(v) *symbols* $\neg$, $\Rightarrow$, $\forall x$ (where x is a variable) ,) , (.

First we define the set of *terms* by requiring it to be the set X with the properties that (i) each individual variable or constant is in X (ii) if $t_1, .., t_k$ are elements of X and $f_{i_k}^k$ is a functional symbol of arity k then $f_{i_k}^k(t_1, t_2, ..., t_k)$ is in X (iii) if Y satisfies (i), (ii) then $X \subseteq Y$.

Next, the set of *formulas* is defined as the set Z with the properties (i) for each predicate symbol $P_{i_k}^k$ and any set $\{t_1, t_2, .., t_k\}$ of terms, the expression $P_{i_k}^k(t_1, t_2, ..., t_k)$ is in Z (ii) for each pair α, β of elements of Z, the expressions $\neg\alpha, \alpha \Rightarrow \beta, \forall x\alpha$ are in Z for each variable x (iii) if Y satisfies (i), (ii) then $Z \subseteq Y$.

The existential quantifier is defined by duality: $\exists x P(x)$ is $\neg\forall x \neg P(x)$. We distinguish between free and bound occurrences of a variable x in a formula α: an occurrence is bound when this occurrence happens in a part of the formula (subformula) preceded by the quantifier sign; otherwise the occurrence is free. It is intuitively clear that a formula in which all occurrences are bound is a proposition i.e. either true or false in a given model.

To define the syntax, one should specify the axioms. Axioms of T can be divided into two groups: the first group consists of axioms of predicate logic, the second group consists of specific axioms (like commutativity in arithmetic); when the second group is present, we speak of a *first order theory*. The axiom schemes of the first group may be chosen as follows:

– (Ax1), (Ax2), (Ax3) are axiom schemes for propositional logic.

– (Ax4) $\forall x \alpha(x) \Rightarrow \alpha(t)$ where x is a variable, t is a term and t contains no variable such that x occurs in a sub–formula quantified with respect to that variable;
– (Ax5) $\forall x(\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \forall x \beta)$ where the variable x is not free in α.

Specific axioms depend on T; for instance the theory of equivalence relation may be expressed by means of a binary predicate symbol P_1^2 and axioms
(1) $\forall x P_1^2(x,x)$;
(2) $\forall x \forall y (P_1^2(x,y) \Rightarrow P_1^2(y,x))$;
(3) $\forall x \forall y \forall z (P_1^2(x,y) \Rightarrow (P_1^2(y,z) \Rightarrow P_1^2(x,z)))$.

The set of *derivation rules* consists of two rules: modus ponens (MP) known from propositional logic and *quantification* (generalization) rule Q which is the binary relation consisting of pairs of the form $(\alpha, \forall x \alpha)$ where x is any variable. A *theorem* of predicate logic is any formula which may be obtained from an instance of an axiom by applying a derivation rule a finite number of times. Semantics of a first order theory T is defined according to Tarski [66] as follows.

A *model* M for the theory T is a pair (D, f) where D is a set and f is an interpretation of T in D i.e. f assigns to each individual constant c an element $f(c) \in D$, to each predicate symbol P_i^k a relation $f(P_i^k)$ on D of arity k and to each functional symbol f_i^k a function $f(f_i^k)$ from D^k to D. Truth of a formula, relative to M, is defined inductively on complexity of the formula. To this end, we consider the states of the model M as sequences $\sigma = (a_i)_i$ of elements of D. Given a formula α and a state σ, we need to declare when the formula α is satisfied by σ, in symbols, $\sigma \models \alpha$. We define a map F_σ which assigns an element of D to each term of T. Individual variables are interpreted via a given state σ: $F_\sigma(x_i) = a_i$. The mapping F_σ is equal to f on individual constants. The inductive condition is as follows: if F_σ is already defined on terms $t_1, t_2, ..., t_k$ and f_i^k is a functional symbol then $F_\sigma(f_i^k(t_1, t_2, ..., t_k)) = f(f_i^k)(F_\sigma(t_1), F_\sigma(t_2), .., F_\sigma(t_k))$.

The satisfiability $\models$ is defined inductively as follows:
(i) $\sigma \models P_i^k(t_1, t_2, ..., t_k)$ if and only if $f(P_i^k)(F_\sigma(t_1), .., F_\sigma(t_k))$ holds;
(ii) $\sigma \models \neg \alpha$ if and only if it is not true that $\sigma \models \alpha$;
(iii) $\sigma \models (\alpha \Rightarrow \beta)$ if and only if $\sigma \models \alpha$ implies that $\sigma \models \beta$;
(iv) $\sigma \models \forall x \alpha(x)$ if and only if $\sigma^x \models \alpha$ for each state σ^x where (letting x to be the variable x_i) σ^x is like σ except that the i-th member of σ^x need not be equal to a_i.

These conditions allow to check for each formula whether it is *satisfied* by a given state. A formula is *true in the model* M if and only if it is satisfied by every state σ. A formula is *true* (*tautology)* if and only if it is true in every model M. Observe that a formula $\alpha(x_1, ..., x_n)$ is true if and only if its closure $\forall x_1 ... \forall x_n \alpha(x_1, ..., x_n)$ is true.

The first order theory PC without specific axioms is called the *predicate calculus*. Properties of first order theories depend on specific axioms so we here recapitulate the facts about the predicate calculus. The soundness of predicate calculus can be easily established by structural induction: each theorem of PC is true as all instances of axiom schemes (Ax1)-(Ax3) are true and truth is preserved by derivation rules MP and Q. The important Gödel completeness

theorem [27], [42] states that predicate calculus is complete: each tautology is a theorem. Decidability problems for first order theories involve questions about the formalizations of the intuitive notion of a *finite procedure* and can be best discussed in the frame of the fundamentally important first order theory of Arithmetic (cf. [42]): a predicate calculus without functional symbols and individual constants is called pure predicate calculus PP while predicate calculus with infinite sets of functional symbols and individual constants is called functional calculus PF. The classical theorem of Church [11], [42] states that both PP, PF are recursively undecidable (algorithmically unsolvable). On the other hand, many problems are recursively decidable (algorithmically solvable) however their time– or space–complexity makes them not feasible e.g. satisfiability problem for propositional calculus is NP-complete [22]. Many problems in logic and applications in KDD are computationally hard: for those problems efficient heuristics have to be found.

B4.4 Deductive systems (DS)

Here we sum up the features of deductive systems like propositional logic or predicate calculus. They are important for KDD because they provide tools for reasoning with knowledge. By a *deductive system*, we understand a tuple $(Ax, Gen, \vdash)$ where Ax is a set of *axioms* (meaning by an axiom a formula which is assumed to be well-formed and desirably true), Gen is a set of *inference* (*derivation*) *rules*, each rule R being a relation on the set of formulas and $\vdash$ is a relation on formulas such that whenever $\Gamma \vdash \alpha$ holds this means that there exists a *formal proof* of α from Γ (α is derivable from Γ) i.e. there exists a finite sequence (the formal proof) $\alpha_1, ..., \alpha_k$ such that (i) α_1 is either an axiom or an element of Γ (ii) α_k is α (iii) each α_i $(i = 2, ..., k)$ is either an axiom or is in Γ or satisfies $R(\alpha_{j_1}, ..., \alpha_{j_m}, \alpha_i)$ for some $R \in Gen$ and a subset $\{\alpha_{j_1}, ..., \alpha_{j_m}\}$ of $\{\alpha_1, ..., \alpha_{i-1}\}$. Any formula α such that $\vdash \alpha$ (meaning $\emptyset \vdash \alpha$) is said to be a *theorem of the deductive system*. From these definitions, the properties of $\vdash$ follow: (a) $\Gamma \subseteq Cn(\Gamma)$ where $Cn(\Gamma) = \{\alpha : \Gamma \vdash \alpha\}$; (b) $Cn(\Gamma) = Cn(Cn(\Gamma))$; (c) $Cn(\Gamma) \subseteq Cn(\Gamma')$ whenever $\Gamma \subseteq \Gamma'$ (the Tarski axioms for syntactic consequence [65]). Semantics of a deductive system is defined with respect to a class of specified structures called models: there exists a mechanism which for each model and each formula assigns to this formula a subset of the model domain (called the *interpretation* of the formula in the model). A formula is true with respect to a given model in case its interpretation in the model equals the model domain. A formula is *true* (is a *tautology*) in case it is true with respect to all models (in the assumed class of models).

The semantic consequence $\models$ (entailment) is defined on sets of formulas as follows: $\Gamma \models \Gamma'$ if for any model M in which each formula in Γ is true, each formula in Γ' is true in M.

Properties of DS: soundness, consistency, completeness, decidability, expressiveness, complexity

Among properties of deductive systems there are some whose importance de-

serves them to be mentioned separately. The first of them is *soundness* (of axiomatization) which means that all theorems of the system are true. The dual property of *completeness* means that each tautology has a formal proof in the system. As a rule verification of soundness is straightforward while the completeness proofs are usually non-trivial. Another important property often intervening in completeness proofs is *consistency*: a set Γ of formulas is consistent if there is no formula α such that both α and its negation are derivable from Γ. Another important question is whether there exists an algorithm which for each formula can *decide* if this formula is a theorem; if yes, we say that the deductive system is decidable. In this case we may ask about the time - , space - complexity of the decidability problem. We may study complexity of other problems like *satisfiability* (whether the interpretation of the formula is non-empty). From the point of view of knowledge representation, it is important to decide what properties can be expressed by means of formulas of the deductive system. A useful meta–rule is that the greater expressibility the greater complexity of problems about the deductive system. These properties are important for KDD; for example, if our reasoning system is complete, it is enough to remember only the axioms and inference rules to represent all theorems about our knowledge.

Resolution and logic programming

It is desirable from computing view point to have systems for automated deduction; the widely accepted technique for this is *resolution* due to J.A.Robinson [14]. It requires *clausal* form of formulas i.e. a conjunction of disjunctions of literals (a literal is a variable or its negation). Symbol like $\{p, q\}$ means a disjunction of literals p, q and a symbol like $\{.\}; \{..\}; ...; \{...\}$ means a conjunctions of disjunctions i.e. a clause. Resolution uses *refutational proof technique*: instead of checking validity of α it checks unsatisfiability of $\neg\alpha$; to this end $\neg\alpha$ is represented in clausal form and the resolution rule:

from clauses $a \cup p$ and $b \cup \neg p$ the clause $a \cup b$ is derived

is applied a finite number of times. Final appearance of the empty clause $\Box$ witnesses unsatisfiability of $\neg\alpha$ hence validity of α. The resolution calculus is sound and complete with respect to entailment $\models$. Resolution in predicate calculus involves unification i.e. the process of finding substitutions making two terms containing free variable identical. For extensions and refinements see [14].

Particularly important from computational point of view is *Horn clausal logic* [32] based on Horn clauses of which *Horn facts* are of the form

$\forall x_1 ... \forall x_m P^k_{i_k}(\tau_1, ..., \tau_k)$

and *Horn rules* are of the form

$\forall x_1 ... \forall x_k \alpha_1(x) \wedge .... \wedge \alpha_n(x) \Rightarrow \beta(x)$.

A set of Horn clauses is a *Horn clausal theory* T. Inferences for T are based on inference rules of the form: $\alpha_1(c) \wedge .... \wedge \alpha_n(c)/\beta(c)$ where c is a ground term i.e. term without variables corresponding to Horn rules in T. A proof $T \vdash \gamma$ is a finite sequence of inferences starting from an inference based on a fact and ending with γ. This calculus is sound and complete. Horn clausal logic can be considered as a generative device for incremental buildup of a set from Horn facts (alphabet)

and Horn rules (generating rules). It has been applied in implementations of PROLOG and DATALOG in particular in logic programming [38].

The idea behind logic programming is that the logic program is a specification written as a formula in a logical language and the inference engine for the construction solution consists of a deduction system for this language. The system of deduction in logic programming is resolution. The logic programs can be of the form known as definite clause programs (a definite clause is a universally quantified disjunction of one or more literals, only one of which is negated). They are executed by adding a goal being a clause in a special form.

Semantics for logic programs can be defined by Herbrand interpretations. A Herbrand interpretation is based on the Herbrand universe i.e. the set of all ground atoms constructed from constants, function and predicate symbols in the program. The least Herbrand model of a logic program can be defined as the least fixed point of a certain function from Herbrand universe into Herbrand universe. Any predicate calculus sentence can be transformed into a set of clauses and next resolution, like in the tableau method, can be used to test, by refutation, the validity of entailment in predicate calculus.

An example of logic programming system is PROLOG (Colmerauer, 1972). More details can be found in [38]. These systems may be regarded as engines for constructing *knowledge bases* from data.

One of the main tasks of inference is to obtain a description of a target object satisfying (exactly or in satisfactory degree) a given specification (formulated in some logical language). In the *constraint programming logic* [68] the construction schemes of such objects can be extracted from logical proofs.

Theorem provers

Automated theorem proving was initiated in 1950's and by 1960 various computer programs for theorem proving were implemented (Newell, Davis and Putnam, Gilmore, Prawitz, Hao Wang) and able to prove very simple theorems. Resolution technique (J.A. Robinson, 1965) proved to be much more powerful and by now most theorem provers use resolution. In spite of progress much still remains to be done in first place in discovering proof strategies. This will need in particular to introduce some similarity measures on proofs. Moreover, KDD stimulates research towards revision of exact formal proofs by introducing instead of them schemes of approximate reasoning extracted from data [55] (see also Section B6).

B4.5 Other Logics

Modal logic

In many applications in KDD, when our knowledge is incomplete or uncertain, e.g. in mining association rules in databases with inconsistent decision [2], we cannot have exact logical statements, but only we may express certain modalities like *property P is possible*.

Modal propositional logics deal with formalizations of phrases like *it is possible that ...*, *it is necessary that ....* These modalities are formally rendered as generalized quantifiers: $[\alpha]$ is read as *it is necessary that* α, $\langle\alpha\rangle$ is read as *it is possible that* α. These operators are related by duality: $\langle\alpha\rangle$ is the shortcut for $\neg[\neg\alpha]$. The syntax of modal calculus is defined over an alphabet much like that of propositional logic: the only addition is the introduction of modal operator symbols $[.]$ and $\langle.\rangle$. The set of formulas of modal logic is defined as the smallest set X such that (i) X contains all propositional variables (ii) with each pair α, β, X contains $\neg\alpha$, $\alpha \Rightarrow \beta$ and $[\alpha]$.

The axiomatics of modal logics depends essentially on properties of necessity which we intuitively deem as desirable; their rendering in axioms leads to various systems of modal calculi.

Semantics of modal logic is defined as the *possible worlds* (*Kripke*) *semantics* [17]. A model for a modal logic system is a triple $M = (W, R, v)$ where W is a collection of *states* (*worlds*) and R is a binary relation on W (called *accessibility relation*); the symbol v denotes a state of the model (a valuation) i.e. the boolean function on the set of all pairs of the form (w, p) where $w \in W$ is a world and p is a propositional variable. The notion of satisfiability $M, v, w \models \alpha$ is defined by structural induction: (i) $M, v, w \models p$ if and only if $v(w, p) = 1$; (ii) $M, v, w \models \neg\alpha$ if and only if it is not true that $M, v, w \models \alpha$; (iii) $M, v, w \models \alpha \Rightarrow \beta$ if and only if either it is not true that $M, v, w \models \alpha$ or it is true that $M, v, w \models \beta$; (iv) $M, v, w \models [\alpha]$ if and only if $M, v, w_1 \models \alpha$ for each world w_1 such that $R(w, w_1)$.

A formula α is *true in the model* M if and only if $M, v, w \models \alpha$ for each world $w \in W$ and every state v; a formula is *true* if and only if it is true in each model.

Recently modal logics play an important role in many theoretical branches of Computer Science and Artificial Intelligence e.g. in formalization of reasoning by groups of intelligent agents [15]; in applicational domain, we may mention hand–written digit recognition [7] where modal formulas are used to express properties discerning between structural objects.

Temporal and dynamic logics

There are some particular contexts in which formulas of modal logic may be specialized and tailored to specific usage's. Let us mention two such cases i.e. temporal as well as dynamic logics. These logics are useful to express changes of knowledge in a changing environment e.g. in geographic information systems (GIS') [13].

In temporal logics, the set W of possible worlds is interpreted as the set of time instants and the accessibility relation R is the precedence in time relation i.e. wRw_1 means that w precedes w_1 in time (in particular, it may happen that $w = w_1$).

Dynamic logic is applied in the context of properties of programs [30]. In this case the set W is the set of states of an abstract computing machine. Given a program P, the modality $[.]_P$ acts on formulas describing states of the machine and its semantics is defined by the accessibility relation R_P which holds on a pair (w, w_1) if and only if an execution of P starting at w terminates at w_1; specifically, a state w satisfies the formula $[\alpha]_P$ if and only if each state w_1 such

that $R_P(w, w_1)$ satisfies α. This means that the state in which P terminates necessarily satisfies α.

Epistemic logics

These are *logics of knowledge and belief*. Logics for reasoning about knowledge, belief, obligations, norms etc. have to deal with statements which are not merely true or false but which are known or believed etc. to be true at a moment; an additional complication is of pragmatic character: knowledge, belief etc. may be relativized to particular intelligent reasoners (agents) hence we may need also to express statements about group or common knowledge, belief etc. Modal logics have proved suitable as a general vehicle for carrying out the task of constructing such logics. These logics are useful in KDD in e.g. tasks of describing problems of distributed/many–agent nature, in building networks of reasoning agents (e.g. belief networks) etc. Epistemic logics for reasoning about knowledge [31], [72], [29] are built as modal logics with a family $K_i : i = 1, 2, ..., n$ of necessity operators, K_i interpreted as the modal operator *the agent i knows that...*. Syntax of such logic is like that of modal propositional logic except for the above family K_i instead of a single [.] modal operator symbol. Formulas are defined as usual, in particular given a formula α, the expression $K_i\alpha$ is a formula, each i. We have therefore formulas like $K_iK_j\alpha$ read as *the agent i knows that the agent j knows that* α etc. Semantics is the usual Kripke semantics of possible worlds except that to accomodate all $K_i's$, a model is now a tuple $M = (W, v, R_1, ..., R_n)$ where R_i is an accessibility relation of K_i i.e. $v, w \models K_i\alpha$ if and only if $v, w' \models \alpha$ for every w' with $R_i(w, w')$. One may want to express also in this logic statements like *every agent in a group G knows that...* or *it is common knowledge among agents in G that...*. This may be done by introducing additional symbols E_G, C_G for each subset $G \subseteq \{1, 2, ..., n\}$, requiring that for each formula α and each G, expressions $E_G\alpha, C_G\alpha$ be formulas and defining semantics of these formulas as follows: $v, w \models E_G\alpha$ if and only if $v, w \models K_i\alpha$ for each $i \in G$ and $v, w \models C_G\alpha$ if and only if $v, w \models K_{i_1}K_{i_2}, ..., K_{i_j}\alpha$ for each sequence $i_1i_2...i_j$ over G. In other words, the accessibility relation for E_G is $\cap\{R_i : i \in G\}$ and the accessibility relation for C_G is the transitive closure of $\{R_i : i \in G\}$. These logics are axiomatized soundly and completely exactly as modal logics of respective types; additional axiom schemes for logics endowed with operators E_G, C_G may be chosen as follows [29]: $E_Gp \leftrightarrow \wedge K_ip$; $C_Gp \leftrightarrow E_G(p \wedge C_Gp)$ along with the additional derivation rule $(p \Rightarrow E_G(\alpha \wedge \beta)) \Rightarrow (\alpha \Rightarrow C_G\beta)$. These logics are decidable (to check validity it is sufficient to examine at most 2^n worlds where n is the formula length).

Non–monotonic logics

In presence of inconsistencies non–monotonicity may arise: a greater set of premises may lead to a smaller set of consequents because of need for revision of our knowledge. Attempts at formal rendering of this phenomenon has led to non–monotonic logics [41]. Non–monotonic reasoning is central for intelligent systems dealing with commonsense reasoning being non–monotonic.

The non-monotonic logics deal with non-monotonic consequence $\vdash_{nm}$; a gen-

eral idea for rendering $\vdash_{nm}$ may be as follows: try to define $\alpha \vdash_{nm} \beta$ as holding when there exists a belief set Γ of formulas, $\alpha \vdash \beta$ and $\alpha \vdash \gamma$ for sufficiently many $\gamma \in \Gamma$.

This idea is realized in various ways: in *default logic* [56], *probabilistic logic* [1], *circumscription* [34], *autoepistemic logic* [36]. Reiter's default logic is built over predicate calculus L by enriching it with inference rules (called *defaults*) of the form $\alpha(x); \beta(x)/\gamma(x)$ where $\alpha(x), \beta(x), \gamma(x)$ are formulas called resp. the precondition, the test condition and the consequent of the default. For a constant a, the default permits to derive $\gamma(a)$ from $\alpha(a)$ under the condition that *not* $\vdash \neg\beta(a)$; we denote this consequence by $\vdash_d$. Formally, a default theory T may be represented as a pair (K, E) where K, a background context, contains rules and E, the evidence set, contains facts. Rules in K are of two kinds: rules of L and defaults D.

Let us consider one example of characterization of non-monotonic inference $\vdash_{K,E}$ from given (K, E). It may be done by a set of postulates of which we mention: (*Defaults*) $p \vdash_d q \in D$ implies $p \vdash_{K,E} q$; (*Deduction*) $\vdash p$ implies $\vdash_{K,E} p$; (*Augmentation*) $\vdash_{K,E} p$ and $\vdash_{K,E} q$ imply $\vdash_{K,E\cup\{p\}} q$; (*Reduction*) $\vdash_{K,E} p$ and $\vdash_{K,E\cup\{p\}} q$ imply $\vdash_{K,E} q$; (*Disjunction*) $\vdash_{K,E\cup\{p\}} r$ and $\vdash_{K,E\cup\{q\}} r$ imply $\vdash_{K,E\cup\{p\vee q\}} r$. As shown in [24], these rules are sound and complete under a probabilistic interpretation of ϵ – entailment [1]. In this interpretation, probability distributions P_K ϵ-consistent with K in the sense of ϵ-entailment i.e. such that $P_K(\alpha) = 1$ for each $\alpha \in L$, $P_K(\beta|\alpha) \geq 1 - \epsilon$ and $P_K(\alpha) \geq 0$ for each rule $\alpha \vdash_d \beta$ in D (where ϵ is a fixed parameter) are considered. A proposition p is ϵ-entailed by T when for each ϵ there exists a δ such that $P_K(p|E) \geq 1 - \epsilon$ for each δ-consistent probability distribution P_K.

One of the main problems for non-monotonic logics is to define for a given set A of sentences a family $E(A)$ of all its *extensions* i.e. sets of sentences acceptable by an intelligent agent as a description of the world. Different formal attempts to solve this problem are known. Some of them are trying to implement the principle called *Closed World Assumption* (facts which are not known are false). Having the set of extensions $E(A)$ one can define the skeptical consequence relation by taking the intersection of all possible extensions of A.

Let us finally observe that e.g. classification problems can be treated as problems of constraints satisfaction with constraints specified by discernibility conditions and some optimality measures (see also Section B6). Two consistent sets describing situations (objects) are satisfying the discernibility relation if their union creates an inconsistent (or inconsistent in some degree) set.

Many valued logics

Yet another treatment of inference was proposed by Jan Łukasiewicz [40] by assigning to propositions other - than *truth* and *falsity* - logical values. In the first 3-valued logic $Ł_3$ propositions were assigned additionally the value $1/2$ (*possible*); the meaning of implication $p \Rightarrow q$ was determined as $min(1, 1-v(p)+v(q))$ where $v(p)$ is the logical value of the proposition p; similarly negation was determined by the condition $v(\neg p) = 1 - v(p)$.

The same formulas were used to define semantics of $n-valued$ Łukasiewicz logic $\mathrm{Ł}_n$ and infinite-valued logic $\mathrm{Ł}_\omega$ (where logical values are rational numbers from the interval [0,1]). A complete axiomatization for these logics was proposed by Wajsberg [60]. Other systems for many-valued logic were proposed by Post, Kleene and others [60], [70].

In general, one may consider e.g. Łukasiewicz implication, negation etc. as interpreted as functions of suitable arity on the interval [0,1] a fortiori truth values may be regarded as real numbers from the interval [0,1]. In real-valued logics, truth-functional definitions of propositional functors rely on real functions on [0,1] in particular on so-called $t-norms$ and $t-co-norms$ which resp. define semantics of conjunction and disjunction. Implications are usually defined by Łukasiewicz or Kleene implications or so-called *residuated implications* and *negations* are interpreted as decreasing idempotent functions on [0,1] [35]. The interest in many-valued logics grew rapidly after introduction of fuzzy logic by Lotfi A. Zadeh (see below).

Constructive approach, Intuitionism

The above logics were developed on classical principles in which one admits non-constructive proofs. This point of view has been contested by many in first place by L.E.J. Brouwer who put forth an idea of intuitionistic logic; in this logic, proof is understood as a effective construction and the inference $\alpha \Rightarrow \beta$ is true when we have a construction which transform any proof of α into a proof of β. The law of excluded middle (that any sentence is either true or false) does not hold as we may have neither any constructive proof of α nor any constructive proof of $\neg\alpha$. Accepting this point of view leads to parallel intuitionistic variants of above logics. However, the notion of *constructive* is vague; it seems that intuitionistic calculi are forms of algorithmic systems [67]. Let us note that topological semantics for intuitionistic logic has been first proposed by Tarski and Stone and another approach has been proposed by Kripke.

B4.6 Inductive reasoning

Inductive reasoning (inference) can be described as an art of hypothesizing a set of premises P for a given set C of consequences in order to satisfy $P \models C$ [43].

In the above scheme, the unknown element is a set P of premises but also the semantic inference $\models$ has to be specified. Contrary to logical deductive semantic consequence, the inductive inference $\models$ is not concerned with absolute truth - preserving but deals with approximations of concepts and along with mechanisms for concept approximations it should also possess mechanisms for generating degrees of closeness between any concept approximated and its approximation. These degrees may be expressed as numerical values or logical expressions. It is hardly expected that the inductive inference $\models$ may be defined abstracting from the specific background knowledge BK i.e. from the applicational context; one should rather expect a variety of inference mechanisms dependent on the context and extracted from data by using appropriate algorithmic tools. This seems to be a challenge for further development of logic.

A general scheme for inductive reasoning may thus consist of a mechanism for primitive concept formation and a mechanism for construction of complex concepts from primitives. All concepts are of approximative character and mechanisms for their construction must rely on some measures of closeness among concepts. It is important to realize that concepts may be defined in various languages and their comparison is effected by imposing measures of closeness on their extensions.

Particular areas of importance for inductive reasoning are Machine Learning, Pattern Recognition and Knowledge Discovery in Data. Various specific approaches have been proposed for inductive (approximative) inference in these areas. In addition some universal paradigms for inductive reasoning like inductive logic programming [46] have been developed e.g. fuzzy inference, rough inference, probabilistic and statistical reasoning. In what follows we will outline the basic ideas of these approaches.

General view: experimental concepts and approximate definability

Background knowledge is often expressed by means of data tables (e.g. training and test examples in Machine Learning, Pattern Recognition, Inductive Logic Programming). These data tables contain positive as well as negative examples for concepts to be learned or recognized; most often, the given examples form a relatively small part of the concept extension so we may not learn these concepts exactly but approximatively only. In constructing approximations to concepts, the choice of a language for concept description (e.g. a language of logical (boolean) formulas) involving the choice of primitive formulas as well as inference mechanisms is very important. Finding a suitable language is itself a challenging problem in scientific discovery. Approximate definability is effected by means of some measure μ of closeness (similarity) on concept extensions; one of oftener applied measures is the Łukasiewicz measure μ_L [39] based on frequency count, $\mu_L(A,B) = card(A \cap B)/card(A)$ where A, B are concept extensions, rediscovered by Machine Learning and KDD communities recently.

Some natural constraints can be put on concept approximations (i) the extension of concept approximation should be consistent with the concept extension or, at least almost consistent i.e. consistent on training examples and having *small* error on test examples; (ii) some minimality (economy) conditions e.g. minimal length of concept description, universality of description or best adaptability. These conditions are applied in Machine Learning, Pattern Recognition and KDD. Satisfying (i) as well as (ii) may be computationally hard (finding a minimal consistent description is NP-hard for propositional logic [3], [45]) hence there are strategies for suboptimal approximations. Hence as a solution to concept approximation problem we obtain as a rule a family of concept descriptions parameterized by languages chosen or strategies applied for particular solutions. Choice of a particular solution may be motivated by ease of adaptivity, computational complexity of procedures of parameter tuning etc. Let us observe that the choice of primitive concepts is actually a choice of an initial model (relational structure) which itself is to be discovered; an essential criterium is its

expresiveness for concept approximations (see also Section B6).

Relationships to Machine Learning, Pattern Recognition, Inductive Logic Programming

To illustrate the relationship of logic to Machine Learning, Pattern Recognition, Inductive Logic Programming, we borrow an example from KDD. A *decision rule* may be expressed in the form: $a_1 = v_1 \wedge a_2 = v_2 \wedge \ \wedge a_k = v_k \Rightarrow d = v$ where $a_1, a_2, ..., a_k$ are features (predicates) used to build formulas expressing approximating concepts and d is a (vector of) feature(s) used in formulas discerning concepts (decisions) approximated. An *association rule* [2], [16] is a decision rule in which the concept defined by the premise of the rule approximates the concept defined by the consequent of the rule in high degree (high confidence) i.e. sufficient fraction of examples satisfying the premise satisfy the consequent as well and there is a sufficient (defined by a set threshold) number of examples supporting both the premise and the consequent. Similar ideas are exploited in Machine Learning and Pattern Recognition for estimating a strength of a rule.

The problem of selecting relevant features [49], [45] involves some searching procedures like discretization, grouping of symbolic values, clusterization, morphological filtering. These procedures may be conveniently written down in propositional logic (see Section B6). These preliminary procedures define primitive concepts (features) for a given problem of concept approximation. This process leads from primitive features (variables) (e.g. real-valued features, pixel-valued features to new intrinsic features (variables) at the gain being a more compact and a more general description of concepts. Another alternative in search for features is to search for hidden features (variables) - possibly better suited for concept approximation - in terms of which one may define (possibly near-to-functionally) the existing features (variables).

Inductive Logic Programming (ILP) [46] bridges Machine Learning to Logic Programming (cf. B4.4). It gives approximate descriptions of relational concepts from given background knowledge and examples in the language of logical programs. Schematically, ILP may be expressed as inductive hypothesis H formation by means of such tools of Logic Programming as *relative least general generalizations, inverse entailment* [48], [47] in order to satisfy the implication $B \wedge H \models E$ where B, E are respectively, background knowledge and examples. ILP systems have been applied to various problem domains [48], [47].

B4.7 Reasoning about Knowledge

In addition to reasoning *with* knowledge i.e. making inferences from a given knowledge base by means of a chosen inference mechanism, reasoning *about* knowledge involves inferences concerning properties of knowledge (e.g. complete/incomplete), fusion of knowledge (e.g. in distributed or many–agent systems), communication and interface among various sources and bases of knowl-

edge (e.g. fuzzy/rough controller). These aspects are very important for KDD (e.g. data mining in the Internet).

As examples of logics reasoning about knowledge we mention *fuzzy logic* [74], [35] (see Section B7), *rough logic* [52] (see also Section B6), *probabilistic logic* [1], [53]. The latter is based on evaluating evidence based probabilities of inferences $p \Rightarrow q$. In these evaluations often one applies *Bayesian reasoning* based on the Bayes formula. Complex inferences may be carried out in semantic networks known as *Bayesian belief networks* i.e. graphical representations of causal relations in a domain.

An example of a logical system dealing with fusion of knowledge is *mereological logic* based on rough inclusion [54] i.e. a predicate $\mu(X, Y)$ which for concepts X, Y returns the value of degree in which X is *a part of* Y. This idea involves necessity of a calculus for fusion of local rough inclusions at any agent as well as for propagation of rough inclusions among agents.

B 4.8 KDD as a logical process

Data mining can be described as searching data for relevant structures (semantic models) and their primitive properties i.e. *patterns*. These relevant constructs are used to discover knowledge. The process of knowledge discovery can be treated as a kind of inference process (classical, commonsense etc.) based on the constructs found in the data mining stage, leading to efficient solutions of tasks like classification, prediction, etc. The solutions provide us with descriptions of concepts of interest having satisfactory quality. The inference process has its own logic: in some cases it may be based on classical logic but in many cases, due to uncertainty, the logic of inference should be extracted from data as schemes for approximate reasoning [54], [74]. In this latter case a very important issue is knowledge granulation and reasoning with information granules [75, 76], [55] making feasible the reasoning process in case of complex problems like spatial reasoning [63] where the perception mechanisms play important role. Finally, let us mention that the inference process should be dynamically adapted to changing data. This causes the necessity to develop adaptive reasoning strategies that tune parameters of logical models (structures) and formulas to induce the optimal (sub-optimal) concept approximations.

KDD faces currently problems related to cognitive and information aspects of perception and reasoning [59]. This certainly stimulates investigations on foundations of logic towards the revision and redefining of its traditional notions [9]. For example, the notion of a proof seems to evolve towards the notion of an approximate scheme of reasoning (extracted from data) due to uncertainty or complexity of search in possible proof space.

Acknowledgement. This work has been supported by the grant No. 8T11C02417 from the State Committee for Scientific Research (KBN) of the Republic of Poland and by the ESPRIT-CRIT 2 project #20288. Andrzej Skowron has also been partially supported by grant of the Wallenberg Foundation and by grant

from the State Committee for Scientific Research (KBN) of the Republic of Poland.

References

1. Adams, E.W.: The Logic of Conditionals, An Application of Probability to Deductive Logic. D. Reidel Publishing Company, Dordrecht, 1975.
2. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings ACM SIGMOD Conference on Management of Data, Washington, 1993, 207-213.
3. Anthony, M., Biggs, N.: Computational Learning Theory. Cambridge University Press, Cambridge, 1992.
4. Åquist, L.: Deontic logic. In: [18], 605-714.
5. Anderson, A.R., Belnap. N.D.: Entailment: The Logic of Relevance and Necessity, Princeton University Press, Vol.1, 1975.
6. Anderson, A.R., Belnap. N.D., Dunn, J.M: Entailment: Vol.2, Oxford University Press, 1992.
7. Bazan, J.G., Nguyen,Son Hung, Nguyen, Tuan Trung, Skowron, A., Stepaniuk, J.: Decision rules synthesis for object classification. in: E. Orlowska (ed.), Incomplete Information: Rough Set Analysis, Physica - Verlag, Heidelberg, 1998, 23–57.
8. van Bentham, J.: Temporal logic. In: [21], 241-350.
9. van Bentham, J.: Logic after the Golden Age. IILC Magazine, December 1999, Institute for Logic, Language and Computation, Amsterdam University, 12.
10. E. F. Codd, A relational model for large shared data banks, Comm. ACM. Vol.13, No 6, 1970, 377-387.
11. Davis, M.: Computability and Unsolvability, Mc Graw-Hill, New York, 1958.
12. Dubois, D., Prade, H.: Belief Change. Vol. 3 in: Gabbay, D.M., Smets Ph. (eds.), Handbook of Defeasible Reasoning and Uncertainty Management Systems, Kluwer Academic Publishers, Dordrecht, 1998.
13. M. J. Egenhofer, R. G. Golledge (eds.): Spatial and Temporal Reasoning in Geographic Information Systems. Oxford University Press, Oxford, 1997.
14. Eisinger, M., Ohlbach, H. J.: Deduction systems based on resolution. In: [19], 183-271.
15. Fagin, R., Halpern. J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press, Cambridge MA, 1995.
16. Fayyad, U., Piatetsky-Shapiro, G. (eds.): Advances in Knowledge Discovery and Data Mining, MIT and AAAI Press, Cambridge MA, 1996.
17. Fitting, M.: Basic modal logic. In: [20], 368-438.
18. Gabbay, D., Guenthner, F. (eds.): Handbook of Philosophical Logic Vol.2, Kluwer Academic Publishers, Dordrecht, 1994.
19. Gabbay, D.M., Hogger, C.J., Robinson, J.A. (eds.): Handbook of Logic in Artificial Intelligence and Logic Programming Vol.1, Oxford University Press, New York, 1993.
20. Gabbay, D.M., Hogger, C.J., Robinson, J.A. (eds.): Handbook of Logic in Artificial Intelligence and Logic Programming vol.3, Oxford University Press, New York, 1993.
21. Gabbay, D.M., Hogger, C.J., Robinson, J.A. (eds.): Handbook of Logic in Artificial Intelligence and Logic Programming Vol.4, Oxford University Press, New York, 1995.

22. Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to the Theory of NP-completeness. W.H. Freeman and Company, New York, 1979.
23. Gärdenfors, P., Rott, H.: Belief revision. In: [21], 35-132.
24. Geffner, H.: Default Reasoning: Causal and Conditional theories. MIT Press, Cambridge, MA, 1992.
25. Gentzen, G.: Untersuchungen über das logische Schliessen, Math. Zeitschrift Vol.39 (1934) 176-210, 405-431.
26. Gonzalez. A.J., Dankel, D.D.: The Engineering of Knowledge Based Systems: Theory and Practice. Prentice Hall, Englewood Cliffs, NJ, 1993.
27. Gödel, K.: die Vollständigkeit der Axiome des Logischen Funktionenkalküls, Monatshefte für Mathematik und Physik Vol. 37 (1930), 349-360.
28. Gödel, K.: Über formal unetscheidbare Sätze der Principia Mathematica und Verwandtersysteme I, Monatshefte für Mathematik und Physik Vol.38 (1931), 173-198.
29. Halpern, J.Y.: Reasoning about knowledge: a survey. In: [21], 1-34.
30. Harel D.: Dynamic logic. In: [18], 497-604.
31. Hintikka, J.: Knowledge and Belief. Cornell University Press, Ithaca, N.Y., 1962.
32. Hodges, W.: Logical features of Horn clauses. In: [19], 449-503.
33. Hughes, G.E., Creswell, M.J.: An Introduction to Modal Logic. Methuen, London, 1968.
34. Konolige, K.: Autoepistemic logic. In: [20], 217-295.
35. Kruse, R., Gebhardt, J., Klawonn, F.: Foundations of Fuzzy Systems, J. Wiley, New York, 1994.
36. Lifschitz, V.: Circumscription. In: [20], 297-352.
37. Leśniewski, S.: On the foundations of mathematics. In: Surma, S., Srzednicki,J.T., Barnett, D.I., Rickey, F.V. (eds), Stanislaw Leśniewski. Collected Works, Kluwer Academic Publishers, Dordrecht (1992), 174-382.
38. Lloyd, J. W.: Foundations of Logic Programming. Springer- Verlag, Berlin, 1984.
39. Łukasiewicz, J.: Die logischen Grundlagen der Wahrscheinchkeitsrechnung. Krakow, 1913.
40. Łukasiewicz, J.: Philosophische Bemerkungen zu mehrwertigen Systemen des Aussagenkalküls, Comptes rendus de la Société des Sciences et des Lettres de Varsovie Vol.23 (1930), 57-77.
41. Makinson, D.: General patterns in non-monotonic reasoning. In: [20], 35-110.
42. Mendelson, E.: Introduction to Mathematical Logic. Van Nostrand, New York, 1960.
43. Michalski, R.: Inferential theory of learning as a conceptual basis for multistrategy, Machine Learning vol. 11 (1993), 111-151.
44. Michalski R., Tecuci G.: Machine Learning. A Multistrategy Approach Vol.4, Morgan Kaufmann, San Francisco, 1994.
45. Michell, T.M.: Machine Learning. Mc Graw-Hill, Portland, 1997.
46. Muggleton, S.: Foundations of Inductive Logic Programming. Prentice Hall, Englewood Cliffs, 1995.
47. Muggleton, S., Learning from positive data, in: Proceedings of the 6th International Workshop on Inductive Logic Programming, Lecture Notes in Artificial Intelligence 1314, Springer Verlag, Berlin, 1997.
48. Muggleton, S., Feng, C., Efficient induction of logical programs, in: Muggleton, S. (ed.), Inductive Logic Programming, Academic Press, New York, 1992.
49. Nadler M., Smith E.P.: Pattern Recognition Engineering, Wiley, New York, 1993.
50. Nute, D.: Conditional logic. In: [18], 387-440.

51. E. Orłowska (ed.): Incomplete Information: Rough Set Analysis. Physica–Verlag, Heidelberg, 1998.
52. Pawlak, Z.: Rough sets – Theoretical Aspects of Reasoning about Data. Kluwer Academic Publishers, Dordrecht, 1991.
53. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, San Francisco, 1988.
54. Polkowski, L., Skowron, A.: Rough mereology: A new paradigm for approximate reasoning. Intern. Journal of Approximate Reasoning Vol.15, No 4 (1996), 333-365.
55. Polkowski, L., Skowron, A. : Towards adaptive calculus of granules. In: [77], Vol.1, 201-227.
56. Poole, D.: Default logic. In: [20], 189-216.
57. Prawitz, D.: Natural deduction, a proof theoretic study. Stockholm Studies in Philosophy Vol.3, Almquist & Wiksell, Stockholm, 1965.
58. Rescher, N.: Many-valued Logics. Mc Graw Hill, New York, 1969.
59. Roddick J.F., Spiliopoulou M.: A Bibliography of Temporal, Spatial, and Temporal Data Mining Research. Newsletter of the Special Interest Group (SIG) on Knowledge Discovery & Data Mining Vol.1, No 1 (1999), 34-38.
60. Rosser, J.B., Turquette, A.R.: Many-valued Logics. North-Holland, Amsterdam, 1952.
61. Russel, S.J., Norvig, P.: Artificial Intelligence. A Modern Approach. Prentice Hall, Englewood Cliffs, 1995.
62. Sandewall, E., Shoham, Y.: Non-monotonic temporal reasoning. In: [21], 439-498.
63. `http://agora.leeds.ac.uk/spacenet/spacenet.html`
64. Stalnaker, R.: A theory of conditionals. In: N. Rescher (ed.), Studies in Logical Theory, Blackwell, Oxford, 1968.
65. Tarski, A.: On the concept of logical consequence. In: Logic, Semantics, Metamathematics, Oxford University Press, Oxford, 1956.
66. Tarski, A.: Der Wahrheitsbegriff in den Formalisierten Sprachen, Studia Philosophica Vol.1 (1936), 261-405.
67. Troelstra, A. S.: Aspects of constructive mathematics. In: Barwise, J. (ed.), Handbook of Mathematical Logic, North Holland, Amsterdam, 1977, 973-1052.
68. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press, London 1993.
69. J. D. Ullman, J. Widom, A First Course in Database Systems, Prentice–Hall, Inc., Englewood Cliffs, 1997.
70. Urquhart, A.: Many-valued logic. In: [20], 71-116.
71. Wille, R.: Formale Begriffsanalyse: Mathematische Grundlagen. Springer-Verlag, Berlin, 1996.
72. Von Wright, G.H.: An Essay in Modal Logic. North Holland, Amsterdam, 1951.
73. Zadeh, L.A.: Fuzzy sets. Information and Control Vol.8 (1965), 333-353.
74. Zadeh, L.A.: A theory of approximate reasoning. In: Hayes, J.E., Michie, D., Mikulich, L.C. (eds.): Machine Intelligence Vol.9, J. Wiley, New York, 1979, 149-194.
75. Zadeh, L.A.: Fuzzy logic = computing with words. IEEE Trans. on Fuzzy Systems Vol.4 (1996), 103-111.
76. Zadeh, L.A: Toward a theory of fuzzy information granulation and its certainty in human reasoning and fuzzy logic. Fuzzy Sets and Systems Vol.90 (1997), 111-127.
77. Zadeh, L.A., Kacprzyk, J. (eds.): Computing with Words in Information/Intelligent Systems Vol. 1-2. Physica-Verlag, Heidelberg, 1999.