REASONING

(Third Draft)

Jürgen Dix

Department of Computer Science

Artificial Intelligence Group,

University of Koblenz-Landau,

Rheinau 1,

D-56075 Koblenz, Germany

March 12, 1998

Overview

Reasoning is the process to deduce (or derive) conclusions from given information. This is a fairly general description and we will try in the sequel to give the reader some more concrete ideas about how particular reasoning systems look like.

We will first describe *classical* reasoning systems, the most famous of which are *propositional* and *first order predicate* logic. These systems form the very basis for many others, like *temporal*-, *modal*-, *deontic*- or *intuitionistic*- logic, just to name a few. We will then point out weaknesses of this classical approach, namely the failure to handle *common-sense knowledge* adequately. We claim that one of the main reasons for this failure is the *monotonicity* of classical logic, which is inherently built-in, i. e. some conclusion, once proven true, is true forever. Consequently we describe *non-monotonic* formalisms to overcome these difficulties. Although the main ideas for such reasoning-systems were already introduced in the eighties, their investigation still is a central field of research with close connections to *AI Languages*, *Deductive Databases*, *Expert Systems*, *Theorem Proving* and *Knowledge Representation*.

1 Introduction

The first serious attempt to understand the general rules of logic dates back to Aristotle (about 330 BC). His famous work *Organon* can be seen as the first systematic treatise of logic. The classical example of a valid deduction is to derive from *"All humans are mortal"* and *"Socrates is a human"* the conclusion *"Socrates is mortal"*. It took almost 2000 years, until G.W. Leibniz renewed the idea of creating a complete system of logic in 1670. Then G. Boole introduced his own ideas (*boolean calculus*) in the middle of the 19th century. Finally G. Frege in his famous *Begriffsschrift (1879)* suceeded in defining a system which led to *first order predicate logic* and contained a complete form of Boole's system called *propositional logic*. Both systems (synonymous with logic for a long time) were investigated extensively in the first part of the 20th century. Three main problems arose.

1. *Syntax:* How to define a precise formal language in which statements can be presented?
2. *Semantics:* How to relate these syntactic expressions to the real world, i.e. how to define their *truth* or *falsity*?
3. *Computability:* To determine a formal calculus which allows us not only to derive just valid statements, but also **all** such *true* conclusions: we want a *complete* system.

Concerning the syntax, various different systems have been developed and are currently in use. In fact, every intelligent system must represent knowledge and data in one way or another (see the articles on *AI LANGUAGES AND PROCESSING*, *EXPERT SYSTEMS*, *KNOWLEDGE MANAGEMENT*). We can roughly distinguish between *propositional* and first-order *predicate* logic. While statements from propositional logic are built from boolean variables $a, b, c, \ldots$ (these variables can become either *true* or *false*) using the connectives $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction) and $\rightarrow$ (implication), first-order predicate logic also allows predicate and function constants as well as the quantifiers $\forall$ and $\exists$ to quantify over individuals. Our example in the beginning can be naturally formulated by $\forall x\,(human(x) \rightarrow mortal(x))$, $human(Socrates)$ and $mortal(Socrates)$. Predicate and function constants can have any specified arity: in our example, *mortal* and *human* are unary predicates, and *Socrates* is a nullary function symbol, i. e. with no argument. Note that this example can not be properly reflected in propositional logic, because there we do not have variables for individuals (like $Socrates$). Rather we have variables representing *true* or *false*. Let $human$ stand for humans, $socrates$ for Socrates and $mortal$ for mortals. We can represent our example as $human \rightarrow mortal$, $socrates \rightarrow human$ and $socrates$. Then we can derive $human$ and $mortal$. But the propositional statement $human \rightarrow mortal$ is quite different from the predicate logic formula $\forall x\ (human(x) \rightarrow mortal(x))$. The first can be seen as an instance of the latter, corresponding to one arbitrary but fixed value of x. Note that to express the predicate logic formula, we need infinitely many propositional formulae, one for each choice of an x. This may be illustrated with the following two formulae

$$\exists x \forall y \quad (person(x) \wedge person(y)) \rightarrow loves(x, y)$$
$$\forall x \exists y \quad (person(x) \wedge person(y)) \rightarrow loves(x, y)$$

where the ordering of the quantifiers is crucial. If we want to express the first formula in propositional logic, then we have to introduce a constant, say c for x but the resulting formula still represents infinitely many propositional formulae, namely all possible instances. Expressing the second formula is even more difficult: we can not just introduce one new constant for y because every y depends on the x.

Semantics as well as calculi for propositional logic are much simpler than those for predicate logic. Semantics has been developed independently by E. Post and L. Wittgenstein, using *truth-tables*. Suppose a statement like $a \vee (c \rightarrow \neg d) \vee \neg a$ is given. The variables a, c, d can take arbitrary truthvalues (*true* or *false*). But an inspection shows that every choice of truthvalues will make the whole statement true (using the wellknown rules of the form: $\neg a$ is true iff[1] a is *false*, $a \wedge b$ is *true* iff both a, b are *true*, $a \vee b$ is *true* iff at least one of a, b is *true*, $a \rightarrow b$ is *true* iff b is *true* or both

[1]from now on we use the term *iff* to stand for *if and only if*.

a and b are *true*). Therefore $a \vee (c \rightarrow \neg d) \vee \neg a$ is called a *tautology*: its truthvalue is always true and does not depend on the truthvalues or its constituents. The semantics for first-order predicate logic is quite more complex and was defined in the 30ies by A. Tarski (see the next section for more details).

Once 1. and 2. above were settled, many different complete calculi have been developed. They all define what it means that a statement φ can be *derived* from a set of statements Φ: namely to find a *derivation* of φ from Φ using *the specified inference rules of the calculus*. We write $\Phi \vdash \varphi$.

Why can't we be happy and satisfied with classical logic? After all it was developed during 2000 years and fits perfectly to mathematics and science? The reason is that classical logic is often much too weak (it does not allow us to derive what we want) but at the same time it is sometimes too strong in that it allows us to derive everything! Therefore classical logic is not adequate to formalize common-sense knowledge and to handle inconsistencies! We will illustrate these two points in the next two subsections.

1.1 Classical Logic is too weak

What are the problems of classical logic with formalizing common-sense? Let us consider a very simple every-day task: reading a timetable for trains or buses. Such a timetable can be seen as a set of facts which can be encoded very easily into classical logic. If there are informations

$$Train\_from\_to\_at(2nd\_street, 4th\_avenue, 8am)$$
$$Train\_from\_to\_at(2nd\_street, 4th\_avenue, 9am)$$
$$Train\_from\_to\_at(2nd\_street, 4th\_avenue, 10am)$$
$$Train\_from\_to\_at(2nd\_street, 4th\_avenue, 11am)$$

humans will also conclude that *there are no trains* between 8am and 9am, and none between 9am and 10 am. But such conclusions can not be derived using classical logic!

Another example is the formalization of *rules of thumb*, i.e. rules that allow for exceptions.

Example 1 (Flying Birds)

The most famous example is a bird usually flies, unless he is an ostrich or sick. *We can formulate this as*

$$bird(x) \wedge \neg ostrich(x) \wedge \neg sick(x) \rightarrow flies(x).$$

Suppose we know that Tweety is a bird: $bird(Tweety)$. Everyday reasoning will allow us to derive that Tweety flies, but logic will not do. In classical logic, you can only apply the rule if you establish that Tweety is not an ostrich and is not sick. But this information is not available nor can it be classically derived!

1.2 Classical Logic is too strong

The reason why logic can be too strong is the *Ex falso quodlibet* principle: *From inconsistent data, we can derive everything!* Bertrand Russell illustrated this principle by deriving from the (inconsistent) statement $6 = 7$ that he is the pope! It goes as follows: Russell and the pope are certainly at most two different persons. But if $6 = 7$ we can subtract 5 from both sides and get $1 = 2$, so that Russell must be identical to the pope! Although this is often treated as an anecdote, it clearly reflects the behaviour of classical logic: inconsistency implies everything. The formal proof of this property is not very different from Russells argument.

The *Ex falso quodlibet* principle is quite strong and for everyday life not well suited.

For example it seems totally ridiculous to derive from one local inconsistency in a large database (say there are two entries, one saying that Mr. X paid his bill and the other that he did not), to derive any statement at all, like *Mr. Webster does not exist* or *Mickey Mouse is president of the USA*. In conclusion, classical logic does not handle well neither incomplete nor contradictory information, a commonly occurring situation in everyday life. This is so because classical logic was designed to deal with perfect, well-defined mathematical objects.

2 Classical Logics

As we will see in this section, classical logics are *monotone*: once we have derived a statement φ from a set of statements Φ, this statement remains true even if new statements Ψ are added to Φ:

$$\Phi \vdash \varphi \text{ implies } (\Phi \cup \Psi) \vdash \varphi.$$

This property is a *conditio sine qua non* for mathematics and sciences: adding new information should only *increase* the set of derivable knowledge, but never decrease it. Once a theorem is true, it should remain true forever, simply because its original derivation is not affected by new axioms. It turns out that this is exactly the property that makes classical logic often too weak and sometimes too strong (see the last section).

Let us consider our example concerning flying birds. If a logic is strong enough to derive $flies(Tweety)$ from $bird(x) \wedge \neg ostrich(x) \wedge \neg sick(x) \rightarrow flies(x)$ and $bird(Tweety)$ (which would be nice from the common-sense character of this rule), what will happen if we learn later that Tweety in fact is an ostrich or is sick? Comparing with the monotonicity property, Φ consists of $bird(Tweety)$ and $bird(x) \wedge \neg ostrich(x) \wedge \neg sick(x) \rightarrow$

$flies(x)$, while $\Psi = \{ostrich(Tweety)\}$ and φ is $flies(Tweety)$. Obviously we want to revise our previous conclusion: knowing that Tweety is an ostrich should prevent us from deriving that Tweety flies! But this means that the monotonicity property cannot hold!

So the main idea is to give up monotonicity in order to build reasoning systems that allow us to derive *more formulae* in the light of *few information*. If we learn later more about the particular situation at hand, we may have to withdraw our previous conclusions: nothing lasts for eternity.

2.1 Propositional Logic

The language of propositional logic contains propositional variables, usually denoted by $p, q, r, v, \ldots$. These can be seen as placeholders for the truthvalues *true* and *false* We also have the connectives mentioned above: $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction) and $\rightarrow$ (implication). $\neg$ is unary, while all others are binary. One also allows the parentheses "(" and ")" for better readability. It is now easy to define by recursion the notion of a *formula of propositional logic*: we simply declare (1) the propositional variables to be formulae, and (2) if φ, ψ are formula, then so are $(\neg\varphi)$, $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, and $(\varphi \rightarrow \psi)$.

Note that up to now, we only talk about formal expressions. What we really want is to give these connectives a *meaning*: we want that the formal symbol $\neg$ corresponds to negation, that $\wedge$ corresponds to conjunction and so on. So the question is: How should we define the *truth* of a formula? Obviously, the truth of a complicated formula depends in general on the truth of its components. We consider the formula $(p \vee q) \rightarrow q$. Intuitively, we would say that this is not true, at least it is not true under all circumstances. Namely if p is true and q is not, the implication $(p \vee q) \rightarrow q$ is considered not to hold (*true* should not imply *false*). Defining the truth of a compound formula can be done recursively. Suppose we have a *valuation* v of all propositonal variables, i.e. a mapping which associates to every variable a truthvalue (*true* or *false*). We want to extend this valuation to a mapping $\bar{v}$ which associates a truthvalue to every compound formula. In addition, we want this valuation $\bar{v}$ to satisfy the following conditions:

$$
\begin{aligned}
\bar{v}(\neg p) &= \begin{cases} \mathit{true} & \text{if } \bar{v}(p) = \mathit{false}, \\ \mathit{false} & \text{if } \bar{v}(p) = \mathit{true}, \end{cases} \\
\bar{v}(\varphi \wedge \gamma) &= \begin{cases} \mathit{true} & \text{if } \bar{v}(\varphi) = \mathit{true} \text{ and } \bar{v}(\gamma) = \mathit{true}, \\ \mathit{false} & \text{otherwise}, \end{cases} \\
\bar{v}(\varphi \vee \gamma) &= \begin{cases} \mathit{true} & \text{if } \bar{v}(\varphi) = \mathit{true} \text{ or } \bar{v}(\gamma) = \mathit{true}, \\ \mathit{false} & \text{otherwise}, \end{cases} \\
\bar{v}(\varphi \rightarrow \gamma) &= \begin{cases} \mathit{false} & \text{if } \bar{v}(\varphi) = \mathit{true} \text{ and } \bar{v}(\gamma) = \mathit{false}, \\ \mathit{true} & \text{otherwise.} \end{cases}
\end{aligned}
$$

Such a valuation $\bar{v}$ is also called a *model* (namely a model not only of its atomic constituents, the propositional variables, but also of all sentences that are true in it) and can be shown to be uniquely determined by v. Coming back to our example above, the model which is uniquely determined by assigning p to be *true* and q to be *false*, we denote it simply by $\{p\}$, is not a model of $(p \vee q) \rightarrow q$.

Now there are formulae that are *true* in all models, i.e. their truthvalue does not depend on a particular valuation. For example $p \vee \neg p$, $p \rightarrow p$ or $(p \wedge q) \rightarrow (q \wedge p)$ are such formulae: they are called *tautologies*.

But what we really want is to define what it means that a formula *follows* from given ones. And, having defined that, we would like to have an algorithm that allows us to *derive*, by syntactical means, formulae from others.

How could such a *formal calculus* look like? We define the following inference rule, known as modus ponens

$$\textbf{(MP)} \quad \frac{\varphi, \quad \varphi \to \psi}{\psi}.$$

This rule allows us to derive a new formula, namely ψ, from given ones, namely φ and $\varphi \to \psi$. But we also need some formulae to begin with, i.e. tautologies. We define the following 3 formulae as axioms Ax:

$$\begin{aligned}
A_1 : &\quad p \to (q \to p), \\
A_2 : &\quad (p \to (q \to r)) \to ((p \to q) \to (p \to r)) \\
A_3 : &\quad (\neg p \to \neg q) \to (q \to p)
\end{aligned}$$

In this calculus, $\vee$ and $\wedge$ do not occur, they are *defined* by $p \vee q \ := \ \neg p \to q$ and $p \wedge q \ := \ \neg(p \to \neg q)$. This means that all occurrences of "$\vee$" and "$\wedge$" are just abbreviations. As a simple example, we try to derive $p \to p$. The axioms A_1 - A_3 are *schemata*, i.e. p, q, r can be instantiated by arbitrary compound formulae: they are nothing but placeholders. MP applied to A_1 and A_2 (where p is substituted for r) gives $\varphi_4 : \ (p \to q) \to (p \to p)$. We can now substitute $p \to p$ for q and obtain

$$\phi_4' : \ (p \to (p \to p)) \to (p \to p).$$

MP can now be applied to φ_4' and A_1' (where p is substituted for q) and results in $p \to p$.

2.2 Basics of Classical Logic

Let us collect some basic facts which are easy to prove and do not only hold for the just introduced propositonal logic, but for all classical reasoning systems. You can see these facts as introducing the most basic terminology in classical logic. We will need and use these notions in the rest of this article.

The main notion in any logic is the *semantical consequence relation* $\models$:

- $\Phi \models \psi$ if, by definition, all models of Φ are also models of ψ. I.e. every model which makes all formulae of Φ *true*, makes also ψ *true*.

Facts and Definitions 1 (Semantics: $\models$)

Given a set of formulae Φ, which is also called a theory*, we denote with*

1. *$MOD(\Phi)$ the set of all models of Φ: $MOD(\Phi) := \{\bar{v} :\ \bar{v}$ makes all formulae from Φ* true*$\}$.*

2. *$Cn(\Phi)$ the set of all consequences of Φ: $Cn(\Phi) := \{\varphi :\ \varphi$ is* true *in all models of $\Phi\}$.*

   Note that $Cn(\Phi)$ is infinite, even if Φ is finite: $\{p, p \wedge p, p \wedge p \wedge p, \ldots\} \subset Cn(\{p\})$.

We also use $\bar{v} \models \varphi$, to express that in the model $\bar{v}$, the formula φ is true *(or holds).*

How can we determine if a formulae φ is not a consequence of Φ? We have to find a counterexample:

$$\Phi \not\models \varphi \quad \textit{iff} \quad \textit{there exists a model of } \Phi \cup \{\neg\varphi\}.$$

We already mentioned the consistency of a set of formulae. More precisely, we call a theory consistent, if there is a model which makes all statements of this theory true*. The following is easy to establish but very fundamental*

$$\Phi \quad \textit{is consistent} \quad \textit{iff} \quad Cn(\Phi) \neq Fml,$$

where Fml denotes the set consisting of all formulae*, i.e. this set is certainly inconsistent: with every formula ϕ also its negation is contained in Fml. This last statement is exactly the* ex falso quodlibet *from the last section.*

Now that we have defined the consequences of a set of formulae Φ, how can we compute them? We already mentioned the method of truth-tables, which is a brute force method: determine for every valuation of the propositional variables the valuation of the whole set of formulae. If for all models of Φ (i.e. those valuations which make all formulae in Φ *true*) we get *true* as the value of the formula φ, then φ follows from Φ. If we find one model of Φ in which the φ is *false*, then φ does not follow from Φ. This method is a semantical one, in that we just evaluate all possibilities. If the set Φ is empty, we call the consequences *tautologies*, i. e. tautologies are true in all models. Such a method is severely restricted to propositional logic, because we have for any formula only finitely many different models: any particular formular contains only finitely many variables, so there are only finitely many models.

The general notion of a *calculus* (which is also well-suited for non-propositional logics) is the fundamental counterpart of the notion of a semantics.

Facts and Definitions 2 (Derivability: $\vdash$)

Suppose we are given a set of inference rules*, which allow us to derive from a given finite set $\varphi_1, \varphi_2, \ldots, \varphi_n$ of formulae another formula ψ:*

$$\frac{\varphi_1, \varphi_2, \ldots, \varphi_n}{\psi}.$$

Modus Ponens (MP) is such a rule with two assumptions. Suppose also that we are given a set of axioms Ax, i.e. a certain set of formulae.

1. *Both sets specify together a* calculus $\mathcal{C}\mathrm{al}$*.*

2. *Now, given a set of formulae Φ, we can define what it means to* derive formulae from Φ using the inference rules*. Namely we start with Φ (these are our* assumptions*), and we try to apply our inference rules to derive a new formula. This new formula can then also be used, because it is contained in our new set of assumptions.*

   Formally, this has to be done recursively using the notion of a proof*. A proof of a formula φ from a set Φ in a certain calculus $\mathcal{C}\mathrm{al}$, is a finite sequence of formulae $\phi_1, \phi_2, \ldots, \phi_n = \varphi$, where every ϕ_i is either contained in Φ, or is an axiom, or is obtained by an inference rule of $\mathcal{C}\mathrm{al}$ applied to some formulae ϕ_j with $j < i$: the ϕ_j have to be derived before ϕ_i.*

3. *We write $\Phi \vdash \varphi$, if φ can be derived from Φ with respect to the calculus under consideration. I.e. if there is a proof of φ from Φ.*

The MP-rule and the set Ax introduced in the last section constitute a *complete calculus for propositional logic*. This means that whenever a formula follows semantically, $\Phi \models \varphi$ as defined above, then it is also derivable in the calculus: $\Phi \vdash \varphi$. And vice versa: all derivable formulae follow semantically. These properties are among the most interesting for any logic and appropriate calculus:

Soundness: A calculus is *sound*, if all derivable formulae are semantically valid: $\Phi \vdash \varphi$ implies $\Phi \models \varphi$.

Completeness: A calculus is *complete*, if all semantically valid formulae are derivable: $\Phi \models \varphi$ implies $\Phi \vdash \varphi$.

2.3 First-Order Predicate Logic

Propositional logic is completely contained in predicate logic. The difference is that now we have variables ranging over arbitrary individuals. In addition, we have function symbols available.

The language of first order predicate logic consists of a set *var* of individual variables: $x, y, z, x_1, x_2, \ldots$, a set *Pred* of predicate symbols, a set *Funct* of function symbols. Let us try to illustrate predicate logic with a running example, the natural numbers $\mathbb{N}$. Variables will range over natural numbers. We use a nullary function symbol, the "0", and a unary function symbol, the "$s(\cdot)$" (meaning *successor*). Also we use two binary functions, "$+$" and "$\times$". Finally, we use two binary predicates, "$=$" and "$\lneqq$". As in every logic, the most interesting notion is that of a formula. But in contrast to propositional logic, we have here also the notion of a *term*, defined recursively as follows.

Definition 1 (Terms)

A variable is a term. If f is a n-ary function symbol and $t_1, t_2, \ldots, t_n$ are terms, then $f(t_1, t_2, \ldots, t_n)$ is also a term. In our running example, $x, s(x), s(s(x)), 0, s(0), s(x+0), s((x \times x) + s(0))$, etc. are terms. Of particular importance are ground *terms: terms, where no variables appear. These are in our example just $0, s(0), s(s(0)), 0+0, s(s(0+s(0)) \times s(s(0)))$, etc.*

Once we have defined terms, we can define formulae. Like in propositional logic, we use the connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$ and, as new objects that will use the terms just defined, the quantifiers $\forall$ ("for every") and $\exists$ ("there exists"). A predicate logic formula is defined recursively as follows. If P is a n-ary predicate symbol and $t_1, t_2, \ldots, t_n$ are terms, then $P(t_1, t_2, \ldots, t_n)$ is a formula (called atomic). If φ, ψ are formula, then so are $(\neg\varphi)$, $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, and $(\varphi \rightarrow \psi)$. Finally, if x is a variable and φ a formula, then so are $(\forall x \varphi)$ and $(\exists x \varphi)$. x is called a *bound* or *non-free* variable.

This means we can now build formulae of the form $\forall x \exists y \; s(x) \lneqq y$, $\neg \exists x s(x) = 0$, or even

$$\neg(\exists x \exists y \exists z \; (0 \lneqq x \wedge 0 \lneqq y \wedge x^3 + y^3 = z^3))$$

which already represents a quite interesting property (the last sentence, F_3, is the simplest instance of Fermat's theorem which has been proved in its full generality only recently). But we are faced with the same problem as in propositional logic: How to define the truth of such formulae? In propositional logic, the truth of a formula was determined by going back to the lowest constituents of a formula, namely the propositional variables. In any model, these variables got a truthvalue. But what is a first-order model? Note that we have to find a counterpart for the terms!

When we look at our example, a nice area to interpret the formulae are the natural numbers $\mathbb{N}$. So let us take the set of natural numbers $\mathbb{N}$ as our universe, and the well known functions $+^{\mathbb{N}}$, $\times^{\mathbb{N}}$ defined on natural numbers. Let us also denote by $0^{\mathbb{N}}$ the zero in $\mathbb{N}$, and $s^{\mathbb{N}}$ be the successor-function, which associates to any numeral n its successor $n+1$. $=$ stands for equality and $\lneqq$ for the usual "less than" relation. Note that we have *interpreted* our formal symbols to concrete objects in the set $\mathbb{N}$. This enables us to define the truth of a sentence φ in $(\mathbb{N}, +^{\mathbb{N}}, \times^{\mathbb{N}}, \lneqq^{\mathbb{N}}, 0^{\mathbb{N}}, s^{\mathbb{N}})$: we say that $\mathbb{N}$ is a model of φ, written

$$(\mathbb{N}, +^{\mathbb{N}}, \times^{\mathbb{N}}, \lneqq^{\mathbb{N}}, 0^{\mathbb{N}}, s^{\mathbb{N}}) \models \varphi,$$

if interpreting the quantifers over the set $\mathbb{N}$ leads to a *true* number theoretic statement. The sentence $\forall x \exists y \; s(x) \lneqq y$ interpreted in $\mathbb{N}$ says: for all numbers n there is a number

m such that the successor of n is strictly less than m. This statement does clearly hold! But note that we can also interpret our symbols $+, \times, \lneqq, \ldots$ totally differently on the set $\mathbb{N}$. Under such an interpretation, statements that were *true* before do not need to hold anymore.

The general notion of $\Phi \models \varphi$ in predicate logic is as defined before: any model of Φ is also a model of φ. So it remains to define what a general model for a first-order language is: it is exactly as described in our example above.

Definition 2 (First-Order Model)

A first-order model consists first of all of a set U, called the universe, which is used to interpret the quantifiers and the variables. They range over all elements of the universe. It also consists of interpretations of all function and predicate symbols. The interpretation of a function symbol is a function over the universe. The interpretation of a predicate symbol is a relation over the universe.

Once all the symbols in the language are interpreted, i.e. their meaning is fixed, any sentence, i.e. a statement without free variables, can be translated into a mathematical statement over the universe. What happens if we interpret our sentence F_3 in the set of real numbers $\Re$? The variables are now interpreted over real numbers and F_3 is a statement about real numbers. Obviously, $\Re \models \neg F_3$.

Again, as explained in the last section, the valid sentences of predicate logic are those that are *true* in *all* models. Like for propositional logic, there exist various calculi for computing such valid formulae: *Gentzen-*, *Hilbert-type-*, *connection-*, *tableau-* and *resolution-* calculi. In particular the latter type of calculi is well suited for automated reasoning (see the article on THEOREM PROVING).

Why is the logical system just described *first-order* logic? The reason is that we can express statements $\forall x$ where x ranges over the universe of a model, x is a place holder for any particular element of the universe. x does not stand for an arbitrary subset of the universe. If we allow this, we would call such a quantifier to be of *second-order*. Consequently, third-order quantifiers would allow to quantify over sets of sets of elements. The next section will explain that for such higher-order logics, no correct, i. e. sound and complete, calculi can exist.

2.4 Feasability and Decidability

Let us consider the complexity of classical logic. The problem to decide if $\Phi \models \varphi$ holds or not (for arbitrary Φ and φ) is known as the *decision problem*: is there an algorithm which takes arbitrary Φ and φ as input and outputs "yes" if φ follows from Φ and "no" otherwise?

For propositional logic we have already mentioned the method of truth tables which is such an algorithm. It is one of the most famous problems in mathematics to show that there can be no better method than just completely enumerating all possibilities and checking each. This is known as the "P≠NP" statement. If it holds, the decision problem is exponential and not polynomial.

For predicate logic, Alan M. Turing showed in 1936 that the decision problem is only *semi-decidable*. There are algorithms that produce *all* consequences of a given set Φ, but if we want to check if a particular sentence φ follows or not, all we can do is to wait. Two possibilities can occur. If the sentence follows, then we will eventually get it. But if it does not follow, we will never know because the algorithm does not stop. So we can *enumerate* all valid sentences of predicate logic, but we cannot decide this set. Note the difference to propositional logic, where not only $Cn(\Phi)$, but also the set $Fml \setminus Cn(\Phi)$ can be enumerated (using e.g. truth tables). Therefore a decision method is simple: in order to determine if a sentence φ follows, just check both enumerations, φ must be contained in either.

We are faced here with a classical dilemma. While predicate logic has greater expressiveness compared to propositional logic, it is far more complex (only semi-decidable). It can be shown (this dates back to the *Principia Mathematica* of Russell and Whitehead) that the whole of mathematics can be formulated as a first-order theory using appropriate axioms. Therefore first-order predicate logic can be seen as the universal language for mathematics.

We end this section with the remark that second-order logic is not even semi-decidable (as well as higher-order logics in general): there is no algorithm that enumerates all *true* formulae. Therefore, not even a *correct* calculus does exist. The reason for this phenomenon is that higher-order logic is very expressive, it can describe the natural numbers (which is a highly undecidable theory) up to isomorphism. Any correct calculus for such a logic would lead immediately to a decision algorithm for the natural numbers, which is impossible as shown by Gödel.

3 Non-Monotonic Formalisms

We have already mentioned in the introduction two famous examples where classical logic is much too weak to draw reasonable conclusions: interpreting a train table and handling partial information. Of course, the train table can also be seen as an instance of the latter.

In fact, there exist two reasoning paradigms in classical logic opposed to classical deduction: *induction* and *abduction*. Induction can be seen as *abstracting* from several particular examples to a general case, i. e inferring general rules from specific data.

Having observed that the sun rises every day up to now, we conclude with some certainty, that the sun will rise every day in the future as well. Or, more abstractly, if we know that $\phi(0)$ holds and we also know that $\phi(n)$ implies $\Phi(n+1)$, then induction allows us to safely conclude for all natural numbers x: $\Phi(x)$. Induction is an important principle underlying machine learning (see the article on Machine Learning).

The principle of *abduction* can be seen as a converse of the Modus ponens, namely

$$\textbf{(Abduction)} \quad \frac{\varphi \rightarrow \psi, \quad \psi}{\varphi}.$$

This rule allows us to derive φ as a *reason* or a *cause* of the observation ψ, if we know that $\varphi \rightarrow \psi$ holds. Abduction is therefore reasoning from effects to causes or explanations.

Induction was already considered by Socrates but more deeply investigated in the 17th century by Francis Bacon (*Novum Organon*). Abduction was introduced and analysed by Charles Sanders Peirce in 1878.

It is quite interesting that classical logic has often be overestimated for its deductive power. For example Conan Doyle's *Sherlock Holmes*, Agatha Christie's *Hercule Poirot* or Edgar Allan Poe's *Auguste Dupin* are famous examples of classical-logic-based agents, solving almost all of their cases by subtle applications of pure logic. But what they are really using (without explicitly knowing) is highly *nonmonotonic* reasoning: they often have to assume a lot of circumstances, without which they cannot derive anything serious. Such unstated assumptions are often *assumed by default*: they are supposed to hold, even if not stated explicitely.

In the real world, in almost all cases we have only partial information about any given situation, and it helps to make assumptions about how things *normally* are in order to carry out further reasoning. For example, if we learn that someone is a doctor, we usually *assume by default* that he or she is over 25 years old, makes a good salary, etc. Without such presumptions, it would be almost impossible to carry out the simplest commonsense reasoning tasks. Of course, defaults are presumptive precisely because they could be wrong, and if we learn that she (to be specific) is a precocious overachiever, we may have to withdraw the assumption that the doctor is over 25 years old.

One of the first and simplest nonmonotonic reasoning systems was the Closed-World Assumption (CWA). It intuitively means that any information not mentioned in a database is taken to be *false*. More precisely, if an instance $P(t)$ of a predicate P is not contained in the database DB, its negation is assumed to hold. The CWA is therefore very near to classical logic:

Definition 3 (Closed World Assumption: CWA)

$$\mathrm{CWA(DB)} = \mathrm{DB} \cup \{\neg P(t) : \ \mathrm{DB} \not\models P(t)\} ,$$

where $P(t)$ is a ground predicate instance.

P can also have three arguments like

$$Train\_from\_to\_at(2nd\_street, 4th\_avenue, Xam),$$

where $X \neq 8$, $X \neq 9$, and $X \neq 10$, are not derivable and therefore the CWA forces to include their negations $\neg Train\_from\_to\_at(2nd\_street, 4th\_avenue, Xam)$. From this enlarged theory, we can derive what we want.

Reasoning with defaults is inherently nonmonotonic because learning more information may force us to retract a conclusion previously drawn. For example in our train table example, it might happen that we learn of additional trains during the rush hour. Thus we have to revise our previous conclusion, that trains start only on the hour. Defaults therefore represent *uncertain* knowledge.

The main disadvantage of the CWA is that it is often too strong, because it is inconsistent (recall that we had the same problem with classical logic). Suppose our database contains the disjunction $Train(2nd, 4th, 8am) \vee Train(2nd, 4th, 9am)$ and some other facts. This might occur if the printing is not well done and we cannot establish if there is a 8 or a 9 in the schedule. Then neither $Train(2nd, 4th, 8am)$ nor $Train(2nd, 4th, 9am)$ can be derived and therefore both $\neg Train(2nd, 4th, 8am)$ and $\neg Train(2nd, 4th, 9am)$ have to be included, which results in an inconsistent theory!

The attempt to formalize nonmonotonic reasoning so that computer programs could use it as part of their reasoning repertoire was begun by John McCarthy in the 1970's, and the early 1980's saw the development of the major nonmonotonic families: *circumscription*, *default logic*, and *modal nonmonotonic logics*. At the same time, proof methods that were clearly nonmonotonic were also being developed: the so-called *Truth Maintenance Systems* and *negation-as-failure* in logic programming and deductive databases. The problem to define appropriate logics is closely related to the overall way of *expressing partial information*. Two main streams have been followed:

- We allow for a new kind of *inference rules*, rather than formulae in our basic language.
- We try to stick to first-order language and introduce special predicates for handling partial knowledge.

We will describe in the next two sections the two most famous nonmonotonic logics:

default logic DL which follows the first idea and Circumscription CIRC, which is based on the second proposal. Both are based on classical logic, but have a greater deductive power.

Before describing particular reasoning systems, let us talk one more time about defaults in general and distinguish between two different ways to block them. We refer to our running Example 1, *birds usually fly* (unless there is reason to question this piece of information), or to our doctors salary *doctors usually have a high salary.* Such defaults are normally true but can be blocked. For example if we know of a particular bird, Fred, that it does not fly, our default should not be applied. In the same way, we may have access to our fictional doctor's salary records and discover that he earns a low salary. Then the presumption of a high salary is contradicted and defeated. These are called *Type I defeaters*: explicit facts that contradict the conclusion will nullify the justification.

A *Type II defeater*, on the other hand, is more subtle. It undermines the *justification* for a default (which will be introduced below) without contradicting its conclusion; the conclusion may still hold, but we cannot use the default to justify it. For example, discovering that the doctor works in a rural area does not contradict the presumption that the doctor earns a high salary, but it does undermine the justification for the default. In our birds example, we may know that the bird at hand is sick. Again it is still possible that it can fly, but the justification is undermined and therefore the default should be blocked.

3.1 Default Logic

The basic idea behind default logic is to have a pool of *default rules* available and to add to a theory as many conclusions of applicable defaults as consistently possible.

Such a default rule is treated as an inference rule and has the form

$$\textit{Default rule:} \quad \frac{\alpha : \beta}{\gamma}.$$

Its interpretation is, intuitively, *if* α *holds, and it is consistent to assume* β*, then derive* γ. α, the prerequisite, β, the justification as well as γ, the conclusion, are first order formulae.

Whereas default rules are intended to express partial information, ordinary first-order formulae are used to state classical knowledge. In our birds example, we can have $bird(Tweety)$ as classical knowledge and

$$\frac{bird(x) : flies(x)}{flies(x)}$$

as a default, expressing *birds usually fly.* The problem comes in when we learn later that Tweety in fact does not fly: we have to withdraw our previous conclusion (a Type I defeater).

A more subtle problem is with $bird(Tweety)$ and $\forall x\,(ostrich(x) \rightarrow bird(x))$ as classical knowledge together with

$$\frac{bird(x) : \neg ostrich(x)}{flies(x)}$$

as a default, expressing *birds usually fly, when they are not ostriches.* Obviously, we want to derive $flies(Tweety)$ but when we learn later that Tweety is an ostrich, we have to withdraw our previous conclusion (a Type II defeater).

The general problem is to define the set of valid conclusions from a default theory $(\mathcal{D}, W)$ where $\mathcal{D}$ is a set of defaults as described above and W is a set of first-order formulae. In classical logic, we defined the closure $Cn(W)$ from a theory W as the set of conclusions of W. In nonmonotonic reasoning, this set is much too weak and it does not take the default rules into account.

The way out of this is to define the notion of an *extension of a default theory.*

Definition 4 (Extensions of Default Theories)

Such an extension E should represent a possible realization of the default theory. We describe some properties we expect an extension E to satisfy.

1. *Since the facts in W represent certain knowledge, we want those facts to be contained in E, i.e., $W \subseteq E$.*

2. *We want E to be deductively closed in the sense of classical logic, i.e. $Cn(E) = E$.*

3. *Whenever possible we want to use defaults to make our knowledge more complete. In other words, all defaults "applicable" to E must have been applied, i.e., if $A{:}B_1, \ldots, B_n/C \in D$, $A \in E$ and for all i $(1 \leq i \leq n)$ $\neg B_i \notin E$, then C in E.*

4. *E must not contain "ungrounded" beliefs, i.e., every formula in E must be derivable from W and the consequents of applied defaults in a noncircular way.*

The first three properties pose no problem. They prevent contradictions arising by virtue of defaults. It is the fourth property, the exclusion of unwanted elements from an extension, that makes the definition we are looking for rather tricky. In fact, the exact definition is given by a complicated fixed-point equality, which is beyond the scope of this article.

The main departure from classical logic is the definition of several extensions of a default theory, as opposed to the single set $Cn(W)$. Any extension can be thought of as representing a whole set of classical models. Therefore the intersection of all extensions is in general much stronger than the intersection of all classical models, which is exactly $Cn(W)$.

For defaults of a special form, like $\frac{b:f}{f}$, extensions always exist. Also the union of different extensions is inconsistent: thus extensions represent incompatible views of the world (which is wanted). Note that in classical logic two models of a theory are always incompatible, because any classical model makes any formula either *true* or *false*. In general, extensions do not have this property. An extension might not decide every formula: there can be ϕ such that neither $\phi \in E$ nor $\neg\phi \in E$. Figure 1 shows the situation as compared with classical logic. Only some subsets of the set of all classical models MOD(W) correspond to extensions. But the formulae true in all such extensions still is greater than Cn(W).

An important point for representing knowledge is that instead of modifying old statements in the light of new information, it would be much better to just add such new information to the database. This is of great importance if our information is stored in large and heterogenous databases. Instead of physically searching for contradicting statements and modifying them, just adding is a much simpler task. In our birds example, it is much better to store once and for all the default

$$\frac{bird(x) : \neg ab(x) \wedge flies(x)}{flies(x)}$$

in our database. Later, if we learn about exceptions to this rule, we just add formulae of the form $ostrich(x) \rightarrow ab(x)$, $sick(x) \rightarrow ab(x)$, $penguin(x) \rightarrow ab(x)$, where $ab(x)$ stands for "x is abnormal for the rule". In this sense, default logic is modular to a certain degree. It is also possible to encode priorities into defaults: a default can be used to block another one. We can therefore express that some defaults are more important than others.

Default logic is a representative of the class of *consistency-based* or *fixed-point* logics.

3.2 Circumscription

The idea behind Circumscription is to stick completely to classical logic but to *restrict the set of models of a theory*. Partial information is encoded by using special predicates, the abnormality-predicates, and models are selected by taking these predicates into account. To be more precise, a partial ordering $\prec$ between models is defined and therefore such logics are called *model preference logics*. Within such logics, the notion of an *abnormality theory* has been developed by J. McCarthy to represent default reasoning.

Defaults are represented with an introduced abnormality predicate: for example, to say that normally birds fly, we would use:

$$\forall x \left(bird(x) \wedge \neg ab_1(x) \rightarrow flies(x) \right). \tag{1}$$

We already used in the previous section such an abnormality predicate. The difference now is that the whole formula is in our object language, it is not an inference rule like in default logic.

The meaning of $ab_1(x)$ is something like "x is abnormal with respect to flying birds." Note that there can be many different kinds of abnormality, and they are indexed according to kind.

Abnormality theories can represent both Type I and Type II defeat. If it is known that *Tweety* doesn't fly (Type I defeat), we can add (to (1)):

$$bird(Tweety) \wedge \neg flies(Tweety) \ . \tag{2}$$

Here the conclusion $ab_1(Tweety)$ follows in all models. For Type II defeat, we simply assert that Tweety is abnormal, without asserting that he doesn't fly.

Given an abnormality theory, classical logic still allows too many models. In our birds example, the model $\mathcal{M}$ where $bird(tweety)$ and $flies(Tweety)$ as well as the model $\mathcal{M}'$ where $bird(tweety)$ and $ab_1(Tweety)$ are regular models. Intuitively, we prefer $\mathcal{M}$ over $\mathcal{M}'$ because there is no evidence that Tweety is abnormal: *as few things as consistently possible, should be abnormal.* Therefore we are not looking at all models, but only at *minimal models.* Figure 2 shows the overall method. Restricting to a subset of all models obviously strengthens the deductive power: more statements are true in minimal models than in all classical models!

Care must be taken in expressing the predicates that need to be minimized. If instead of $\neg ab(x)$ we used $normal(x)$, minimization would prefer models with less normal entities. This means cicumscription is not symmetric with respect to negation.

Definition 5 (Circumscription)
Circumscription of a theory T which contains abnormality predicates declares to be true all formulae that hold in all models that are minimal with respect to the abnormality predicates.

We are not giving a general definition of minimal models, but rather illustrate it on the following example. Let our theory consist of

$$\begin{array}{lcl} bird(x) \wedge \neg ab_1(x) & \rightarrow & flies(x) \\ ostrich(x) & \rightarrow & ab_1(x) \\ ostrich(x) \wedge \neg ab_2(x) & \rightarrow & \neg flies(x) \\ bird(Tweety) & & \end{array}$$

Intuitively, birds usually fly while ostriches do not. We have to minimize ab_1, ab_2 and get the following two classical models: $\mathcal{M}_1$, where $\neg ab_1(Tweety)$, $bird(Tweety)$ and $flies(Tweety)$ holds. And $\mathcal{M}_2$, where $ab_1(Tweety)$, $bird(Tweety)$ and $\neg flies(Tweety)$ holds. Note the important fact that in order to minimize ab_1, other predicates have to be modified (in our case $flies$), because all models still have to be models of the underlying theory: in our model $\mathcal{M}_2$, we can not just change $ab_1(Tweety)$ into $\neg ab_1(Tweety)$, because $\{\neg ab_1(Tweety), bird(Tweety), \neg flies(Tweety)\}$ is not a model of the underlying theory (the first axiom is not satisfied)!

Therefore in order to minimize certain predicates, otheres have to be allowed to vary during the minimization. The formal definition of minimality takes this into account.

3.3 General Properties

In the previous sections, we studied two particular nonmonotonic inference operations. With this background, we can also step back and consider the properties of such a logic more abstractly, by examining the *metatheoretic* properties of nonmonotonic inference relations. This has been done with two points in mind:

(1) To classify nonmonotonic formalisms. There have been hints that some logics are "stronger" than others in their nonmonotonicity; the metatheory is a way of

providing a more precise characterization of this statement.

(2) To design nonmonotonic logics in a "top-down" fashion. Rather than using a model-restriction operator, we can think of specifying a nonmonotonic logic from above, by constraining the inference operator to have certain desirable properties.

As an example of (2), there is the property of *Cautious Monotony*:

$$\text{If } \Phi \mid\sim \beta \text{ and } \Phi \mid\sim \gamma \text{ then } \Phi \cup \{\beta\} \mid\sim \gamma.$$

$\Phi \mid\sim \varphi$ stands in default logic for φ *is* true *in all extensions of* Φ and in Circumscription for φ *is* true *in all minimal models of* Φ. Of course, any other nonmonotonic logic will induce such a consequence relation $\mid\sim$. Cautious monotony is a restricted version of the monotony condition of classical logic. While default logic (in general consistency-based logics) does not satisfy it, Circumscription does (like all model-prefence logics).

Various other abstract properties have been defined and the induced consequence relations have been investigated (see [Mak94]).

3.4 Other Approaches

There are lots of variants of Default logic and Circumscription as well as approaches more or less related to them. A complete or detailed overview is clearly beyond the scope of this article. But we want to list some systems that we consider worth mentioning.

Description-Logics: These are logics where the underlying language is restricted in such a way, that the induced consequence relation is still decidable or even has good computational behaviour (non-exponential).

Modal nonmonotonic sytems: See the article on *Belief Maintenance*. Such systems have an operator in their language with the meaning *a formula is believed*. Thus we can form statements saying that we believe some formulae and disbelieve others. The main problem is to define an analogue of extensions in default logic. In modal systems these constructs are called *expansions* and are defined by fixed-point equations. These systems are also used to model the notion of *knowledge* as opposed to *belief*.

Probabilistic Systems: See the article on *probabilistic Logic*. Here, probability distributions play a major role. Statements as well as inference rules are true with certain probabilities. Applying inference rules to such formulae means to derive new formulae with appropriate probabilities. One of the problems is to assign the *right* probabilities in advance or to develop a calculus that is robust against small modifications of the associated probabilities. Some people tried to show that nonmonotonic reasoning systems can be seen as the limit of probabilistic systems where probabilities converge to 0 and 1.

Fuzzy logic: See the article on *Fuzzy Logic*. This is not a nonmonotonic reasoning system and we just mention it to remove all doubts about that. Fuzzy logic does not handle *uncertainty*, but rather is well-suited for handling continous-valued variables (as opposed to discrete ones). It has already done its way to industrial applications. Nevertheless there is a nonmonotonic version of fuzzy logic, called *Possibilistic Logic*.

Logic Programming: Here we refer to the articles on *Deductive Databases*, *AI Languages and Processing*. Logic Programming is nonmonotonic, in particular if we use the negation-as-failure mechanism. Semantics for programs with negation are very closely related to classical nonmonotonic reasoning systems and therefore can be used as base-calculi to implement such systems (see [DFN98, AP96]).

Temporal Logic: See the article on *Temporal Logic*. The logics we have discussed sofar are quite *static*, i. e. they do not contain time. However, in everyday life the knowledge changes with time and has to be updated eventually. Temporal logics try to model the dependency of predicates on the time t. They allow to formulate and reason with assumptions about the future and the past. This is especially important for program verfification, where the datatypes depend heavily on the current program state, which is determined by the actual timepoint.

Spatial Reasoning: See the article on *Spatial Databases*. Spatial reasoning deals with the problem to represent and reason with spatial entities of higher dimensions, without resorting to traditional (quantitative) techniques prevalent in computer graphics or computer vision. It is strongly related to *qualitative reasoning*, where one seeks to represent not only commonsense knowledge about the world, but also the underlying abstractions used by engineers and scientists. There are various applications for qualitative spatial reasoning. A particular promising area is *Geographical Information Systems*. For a nice and very readable overview article, also for the non-expert we refer to [Coh97].

Reasoning about Action: Quite early in the history of non-monotonic reasoning, the problem to represent actions and their effects on the current state of the world was recognized as a serious and very difficult one. Various systems were proposed, the most influential among them being the *situation calculus*, which is based on first order predicate logic. The idea is to augment every predicate $At(Fred, School)$ with an additional argument, representing the current state:

$At(Fred, School, s)$. Actions then describe how the predicates change from one situation to the other.

It turned out that all these systems have serious difficulties with at least one of the following, by now classical, knowledge representation problems:

Frame-Problem: Actions in general modify only a few things. But a lot of axioms are needed to describe invariant properties. How to get rid of this huge set?

Qualification-Problem: The enumeration of all conditions under which an action is successful is often unfeasible (there are too many and often they are not known in advance).

Ramification-Problem: How should implicit consequences of actions be handled?

Reasoning about action addresses these problems and is related to *causality*: what actions are *caused* by others, in contrast to simply be *implied* by them. `http://cs.utep.edu/actions/researchers.html` is an actively maintained web page containing more information.

Case-Based Reasoning: Case-Based Reasoning arose at the end of the eigthies as a computational paradigm in which an artificial problem solver finds solutions to new problems by adapting solutions that were used to solve old problems. A case-based reasoner has a case library and each case describes a problem and a solution to that problem. The reasoner solves new problems by adapting relevant cases from the library. Case based reasoning systems are often built using methods from traditional engineering and software technology. They are used for diagnosis, decision support, configurations and planning. We refer to http://www-cia.mty.itesm.mx/~lgarrido/CBR/cbr.html which is actively maintained and contains various pointers to related web-sites.

4 Conclusion

The field of nonmonotonic reasoning started with the goal to model what John McCarthy and others called *jumping to conclusions*. The overall aim was to strengthen the deductive power of classical reasoning systems in order to handle partial and uncertain information. Today, almost 20 years after nonmonotonic reasoning was established as an important research topic, we have made considerable progress in the theoretical understanding of default reasoning. On the other hand, a satisfactory account of the computational properties of human commonsense reasoning still seems to be lacking. Basically, the field has concentrated to a large extent on determining what defeasible conclusions are, but it has been less successful in answering the question what *jumping* to such conclusions means.

Any general reasoning system (as opposed to special-purpose systems) needs to know a lot about how the world looks like. Enormous amounts of rather trivial information have to be handled and it is to be expected the non-monotonic systems can be of use. Doug Lenat's *CYC Project* aims at putting together such common-sense knowledge in a large database that can be accessed by reasoning systems. We refer to http://www.cyc.com/ for a description of this project.

The complexity analysis of nonmonotonic formalisms shows that their computational behavior is much worse than that of corresponding monotonic formalisms. This does not necessarily mean that monotonic formalisms should be used wherever possible: nonmonotonic formalisms often allow us to describe problems in much more compact ways. There are even examples where nonmonotonic problem descriptions are exponentially smaller than any monotonic formulations of the same problem [GPSK95, CDSS97].

Nevertheless, it is generally felt that future research in the field should put more emphasis on computational aspects. Implementations of nonmonotonic systems are on their way. Most of them are closely related with logic programming systems. We refer to the conference series LP & NMR, which also contain references to particular implementations (see also [DFN98]).

In practice, compared with genuine non-monotonic systems, much more prototypes based on probabilistic and fuzzy logic, as well as on case-based reasoning are at work. They all contain non-monotonic components.

5 Literature

For more detailed overview-articles on classical and non-classical logics, we refer the interested reader to the following handbooks: *Handbook of Logic in Artificial Intelligence and Logic Programming* [GHR99], *Handbook of Logic in Computer Science* [AGM99], *Handbook of Philosophical Logic, 2nd Edition* [GG99] and *Handbook of Automated Reasoning* [Rob98]. All contain up-to-date information on the treated subject. In particular, we suggest [Gab94], [Gin94], [Mak94], [RS92], [BD98], [DFN98]. [BDK97] is a recent book on Nonmonotonic Reasoning and treats most aspects on an post-graduate or PhD-level.

References

[AGM99] S. Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors. *Handbook of Logic in Computer Science*, volume 1-6. Oxford University Press, 1992–1999.

[AP96] Jose Julio Alferes and Luiz Moniz Pereira, editors. *Reasoning with Logic Programming*, LNAI 1111, Berlin, 1996. Springer.

[BD98] Gerhard Brewka and Jürgen Dix. Knowledge representation with extended logic programs. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, 2nd Edition, Volume 6, Methodologies*, chapter 6. Reidel Publ., 1998.

[BDK97] Gerd Brewka, Jürgen Dix, and Kurt Konolige. *Nonmonotonic Reasoning: An Overview.* CSLI Lecture Notes 73. CSLI Publications, Stanford, CA, 1997.

[CDSS97] Marco Cadoli, Francesco M. Donini, Marco Schaerf, and Riccardo Silvestri. On compact representations of propositional circumscription. *Theoretical Computer Science*, 182:183–202, 1997. (Extended abstract appeared in: On Compact Representations of Propositional Circumscription. *STACS '95*, pages 205–216, 1995.).

[Coh97] A. G. Cohn. Qualitative Spatial Representation and Reasoning Techniques. In G. Brewka, C. Habel, and B. Nebel, editors, *Proceedings of the 21th German Annual Conference on Artificial Intelligence (KI '97), Freiburg, Germany*, LNAI 1303, pages 1—30, Berlin, 1997. Springer.

[DFN98] Jürgen Dix, Ulrich Furbach, and Ilkka Niemelä. Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations. In Andrei Voronkov and Alan Robinson, editors, *Handbook of Automated Reasoning*. Elsevier-Science-Press, to appear 1998.

[Gab94] Dov Gabbay. Classical vs non-classical logics (the universality of classical logic). In D. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming Vol. 2, Deduction Methodologies*, chapter 6, pages 359–500. Oxford University Press, 1994.

[GG99] Dov. M. Gabbay and F. Guenthner, editors. *Handbook of Philosophical Logic, 2nd Edition*, volume 1–9. Reidel Publ. Co., 1999.

[GHR99] Dov Gabbay, C. J. Hogger, and J. A. Robinson, editors. *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1-6. Oxford University Press, 1993–1999.

[Gin94] Matthew L. Ginsberg. AI and Nonmonotonic Reasoning. In D. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming Vol. 3, Nonmonotonic and Uncertain Reasoning*, chapter 2, pages 1–34. Oxford University Press, 1994.

[GPSK95] Goran Gogic, Christos Papadimitriou, Bart Selman, and Henry Kautz. The Comparative Linguistics of Knowledge Representation. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 862–869, Montreal, Canada, August 1995. Morgan Kaufmann Publishers.

[Mak94] David Makinson. General Patterns in Nonmonotonic Reasoning. In D. Gabbay, C.J. Hogger, and J.A. Robinson, editors, *Handbook of Logic in Artificial*

Intelligence and Logic Programming Vol. 3, Nonmonotonic and Uncertain Reasoning, chapter 3, pages 35–110. Oxford University Press, 1994.

[Rob98] A. Robinson, J. A. and Voronkov, editor. *Handbook of Automated Reasoning*, volume 1-2. Elsevier Science, 1998.

[RS92] Mark Ryan and Martin Sadler. Valuation systems and consequence relations. In *Handbook of Logic in Computer Science, Volume 1*, chapter 1, pages 1–78. Oxford University Press, 1992.

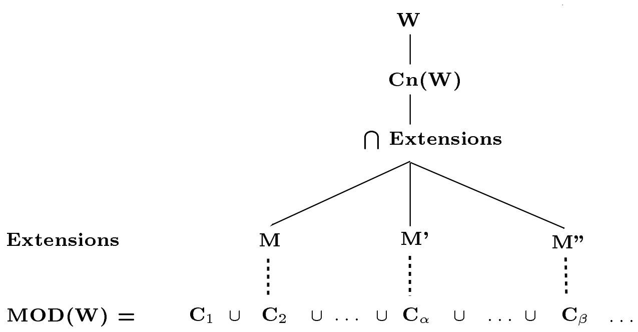

Figure 1: Extensions of Default Theory. This figure shows very clearly, that default reasoning is much stronger than classical reasoning. While classical logic provides only one *closure*, namely $Cn(W)$, for a set of axioms W, default logic selects only some of the many classical models $C_1, \ldots, C_\alpha, \ldots$. These selected models are called *extensions* and reflect the most likely states of the world, given the axioms W.

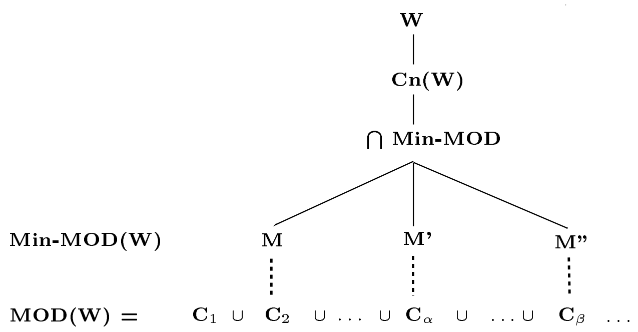

Figure 2: Minimal Models in Circumscription. This figure shows very clearly, that reasoning with minimal models is much stronger than classical reasoning. This is because only a subset of all models of a theory W are minimal. These minimal models play in circumscription the same role as the extensions in default logic.

Crossreferences

- Deduction, see REASONING
- Inferences, see REASONING
- Logic, see REASONING

Related Articles

1. AI LANGUAGES AND PROCESSING
2. BELIEF MAINTENANCE
3. DATABASE LANGUAGES
4. DEDUCTIVE DATABASES
5. EXPERT SYSTEMS
6. TEMPORAL DATABASES
7. THEOREM PROVING
8. RELATIONAL DATABASES
9. DECIDABILITY
10. FORMAL LOGIC
11. FUZZY LOGIC
12. PROBABILISTIC LOGIC