

Bruno Buchberger

**Logic for  
Computer Science**



## 1. Motivation

There are two main sources of motivation for studying logic, the traditional motivation by the foundational questions of *mathematics*, and the recent motivation by the practical needs of *computer science*.

The motivation for studying logic in the context of *mathematics* is based on the observation that reasoning about what one is doing is *useful* for improving the efficiency and quality of one's actions and it may be even *necessary* when in the course of action one is stuck at a point from which, at first sight, there seems to be no escape. *Logic is reasoning about reasoning*. It is *useful* for improving the quality of reasoning and it became an absolute *necessity* when reasoning was stuck at apparently unresolvable paradoxes and contradictions, specifically, in the last century.

The motivation for studying logic in the context of *computer science* is based on the insight that, essentially, automation of problem solving on computers is automation of reasoning and, hence, logic is the key technique for advances in our computer-based problem solving capability. This insight may be gained by a thorough analysis of the past "history" of computer science and may be used for an extrapolation of future trends. We expect that mathematics and computer science will more and more be seen as just the one science of automated intellectual problem solving and logic is the underlying domain-independent technique.

We expand these ideas in some more detail in the subsequent sections. For this we have to start with an informal explication about the nature of mathematics and (mathematical) logic.

### 1.1 Mathematics, Reasoning, Logic

#### 1.1.1 What is Mathematics?

One may draw a dynamic and a static picture of mathematics. In a dynamic picture, *mathematics is the technique of problem solving in models*. In a static picture, *mathematics is the technique of obtaining information in models*.

In both pictures, working in models is the characteristic feature of mathematics. In fact the two pictures are only the two sides of one coin. Problem solving depends crucially on obtaining useful new information and, conversely, obtaining new information must be directed towards "goals" (resolution of problems) in order to avoid useless combinatorial explosion.

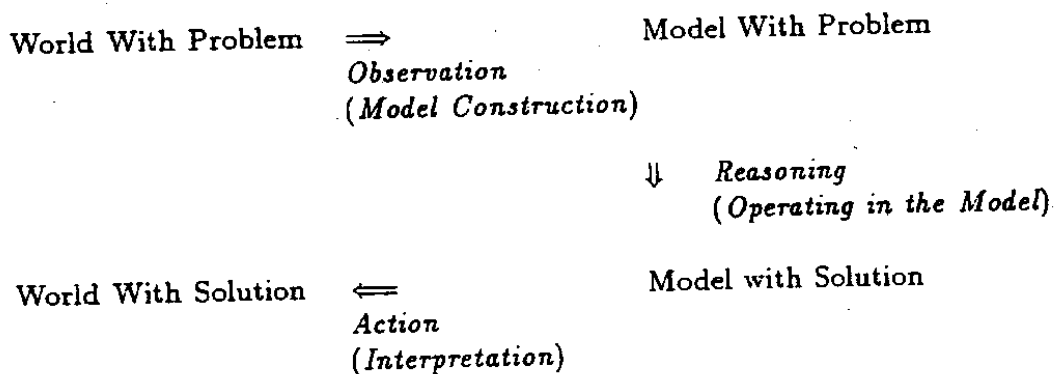
Traditionally, i.e. in the tradition of "pure mathematics" over the last fifty years ("*Bourbakism*"), the *static* picture of mathematics was emphasized. In this introduction, we start from the *dynamic* picture of mathematics because

we feel that this picture is ancient and modern at the same time. In human history, the first mathematical ideas seem to have emerged from the necessity of simple technical problem solving (e.g. the problem of constructing big buildings like the pyramids) and in the present time the computational power of modern computers leads us to a renaissance of the problem solving aspect in mathematics.

The human *intellect* (mind), refined and extended by external means like pencil and paper, computer memory and processors, is the sand-table<sup>®</sup> on which models can be built and explored. Exploring intellectual models is called "*reasoning*" ("deduction", "inference").

\* Sandkasten

Reasoning is at the heart of technical (intellectual) problem solving. A *problem* is a situation in which a desired object is not immediately available. A solution to a problem is an action that produces the desired object. Problem solving based on mathematics (i.e. on reasoning in intellectual models) has three steps:



The first step is the construction of models by *observation*. For observing, *the senses* (eye, ear, ...), often refined by instruments (microscope, earphone, ...), must be brought into contact with the objects. Refined observation in various areas of the real world is the realm of the *natural sciences*. A given problem in real world is reflected by a model problem in the model.

The second step is *reasoning* in the model. This is the proper realm of *mathematics*. By reasoning, new features (statements, insights) of a model can be constructed from known features without involving the senses. The instrument for reasoning in models is the mind whose physical carrier is *the brain*, often refined by instruments (pencil, paper, computer, ...). By reasoning, a solution to the model problem, i.e. an action in the model that produces the desired object in the model, may be found.

The third step is *action*, i.e. interference with the outside world ("interpretation", "realization") for materializing the solution found in the model in order to establish a solution to the original problem. Oversimplifying, one could say that action [based on observation and reasoning] is the realm of the *technical sciences*. The original instrument for transmitting model solutions to real world solutions is *the hand*. In modern times this instrument is refined by "machines".

(Depending on the situation, the view of natural sciences, mathematics, and technical sciences can be broader. Sometimes, one may want to include model

construction and interpretation into mathematics. Sometimes one will prefer natural sciences to encompass reasoning and interpretation. Also, technical sciences may be viewed as embracing observation and reasoning. For the sake of clarity, we oversimplified the situation in our explication above. Thus, the above explication should clarify the specific flavor of natural sciences, mathematics, technical sciences and their mutual distinction.)

The technique of reasoning in models has matured in the evolution of life from the zero stage in primitive creatures where observation and reaction is "spontaneous" (with no "model" of the outside world acting as a buffer and playing-ground) through many intermediate stages to the present stage of human/computer reasoning in extremely complicated models.

### 1.1.2 Fundamental Properties of Reasoning

Reasoning is

- 1. abstract
  - 2. correct
  - 3. ~~controllable~~ <sup>verifizierbar</sup>
  - 4. general.
- (abstrakt)  
 (korrekt) ①  
 (verifizierbar) ②  
 (universell)

#### 1.1.2.1 Reasoning Is Abstract

Reasoning proceeds totally within intellectual models. By definition, no contact or interaction with the "outside world" is taking place during reasoning. We say, "reasoning is *abstract*" (Latin "abstrahere" = pulling away (from the outside world)) or also "formal".

Paradoxically, *the abstractness of reasoning is the source of its practical usefulness*, or, as some people put it, "there is nothing more practical than a good theory". Problem solving by reasoning in models is a dramatic advantage over "spontaneous" problem solving by trial and error (immediate reaction and observation of results) in the outside world. Interaction with the outside world by action and observation can consume a lot of energy, may be quite slow and even can produce irreversible negative effects on the outside world. By contrast, reasoning needs little energy, is fast and reversible. Only the best solution gained by reasoning in the model will be used as the basis for action in the outside world.

However, *the abstractness of reasoning is also its weakness*. Since reasoning is cut off the outside world, it runs risk to "lose contact with reality". Furthermore, models are always simplified views that concentrate on a few interesting aspects of "real world" and forget about other aspects. Hence, a problem solution based on reasoning in a given model, while optimizing one aspect of real world, may well lead to undesirable side-effects with respect to other aspects. These negative effects of technical problem solving based on reasoning have lead to the technological crisis mankind faces today.

When we say that reasoning is abstract we are describing the final outcome of the mathematical problem solving activity. Classically, the outcome is a

① *Verifizierbarkeit ist ein Kriterium für die Objektivität von Aussagen.*

complete proof for a new "theorem" (knowledge) or an algorithm ("process"). Of course, we are aware that, in the creative process of *finding* proofs, looking backward to the special situation from which the model was abstracted, is one of the most important-heuristic techniques. Anyway, in mathematics, an idea obtained from "observation" must be cross-checked by abstract reasoning in the above sense.

verifiable (Verifizierbar)

### 1.1.2.2 Reasoning Must Be Controllable

Reasoning in models must proceed in steps. Each step should be *controllable* in the sense that the step is so simple that "anyone" could repeat it and, starting from the same initial situation, would arrive at the same result. This vague notion of "controllability" today can be made very precise by saying that we require the individual steps of reasoning to be so simple that even a computer can be programmed to execute or check them.

Controllable patterns for steps in intellectual reasoning are called *rules of inference*. In fact, the evolution of intellectual reasoning over the centuries has led to a number of quite general, albeit simple, such rules. These rules form an extremely powerful body of techniques by which intellectual problem solving can be made "intersubjectively controllable". The creative part remaining is finding the right sequence of inferences that leads to a desired knowledge (in the static picture of mathematics) or the desired problem solution (in the dynamic picture of mathematics).

In terms of language, an inference rule is an *algorithm* (mechanism, procedure) that receives a statement as an input and produces a new statement by parsing the given statement and composing the new statement by composing the constituents of the given statement in a new way. For example,

not for all  $x (A) \rightarrow$  there exists  $x$  such that (not  $A$ )

is an inference rule that allows us to take any statement of the form not for all  $x (A)$ , to decompose it into  $x$  and  $A$ , and to compose the new statement there exists  $x$  such that (not  $A$ ).

"Controllability" on the side of reasoning has a counterpart on the side of observation. It is the objective of the natural sciences to establish observations that can be confirmed by "anyone" when applying the same means.<sup>①</sup>

In fact, the combination of "intersubjectively controllable" ("objective") observation and "intersubjectively controllable" reasoning, over the centuries has evolved as the supreme ideal of rational thinking in particular in the "Western", technically oriented, culture. Starting from facts whose truth can be determined by intersubjectively controllable observations and proceeding to new facts whose truth is inferred by intersubjectively controllable reasoning is the basic rhythm of intellectual activity. It is this interplay between controllable observation and reasoning that is called "science" in the Western tradition.

<sup>①</sup> In experiment and in fact it can be proved by repeated observations without exception.

### 1.1.2.3 Reasoning Must be Correct

In order to make this technique of combined observation and reasoning useful for real world problem solving, reasoning must have one more fundamental property, namely “*correctness*”. This means that the inference rules must be such that, whenever a statement  $B$  is inferred from a statement  $A$  by the application of an inference rule, and  $A$  is a true description of a situation in the reality considered (in the “universe of discourse”), then  $B$  should also be a true description of a situation in the same universe. Hence, if the initial statements in a reasoning sequence are true in a given universe and all inference rules applied in the sequence are correct then it is clear that the resulting statement will also be true in the given universe. Briefly, one says that by the inference rules “truth must be transported” from the initial statements to the statements inferred.

The truth of the initial statements may have been determined by observation. For mathematics, this is not relevant. Mathematics is a “relative” science. It is only concerned with deriving new facts from given facts by correct reasoning or, stated differently, mathematics is concerned with determining truth relative to initial statements. Mathematics is not concerned with the determination of the truth of the initial statements.

How can the correctness of inference rules be established? The *evolutionary approach* explains the correctness of (our usual set of) inference rules, for example in in predicate logic, as an accumulation of experience. For example, the correctness of the inference rule

not for all  $x$  ( $A$ )  $\rightarrow$  there exists  $x$  such that (not  $A$ )

is observed by any individual in hundreds of situations in changing “universes”. (Universe:= all dogs;  $A := x$  is black; we observe not for all  $x$  ( $x$  is black), we also observe there exists  $x$  such that (not  $x$  is black). Universe:= all natural numbers;  $A := x$  is divisible by three; we observe (or prove) not for all  $x$  ( $x$  is divisible by three); we also observe there exists  $x$  such that (not  $x$  is divisible by three). Etc.). After many observations (and accumulating the experience over generations through education), we rely on the correctness of the rule. In this evolutionary view, reasoning and, in particular, mathematics can also be conceived as the accumulated and condensed experience of mankind in observing (finite) universes.

Alternatively, in mathematical logic, for establishing the correctness of inference rules, one presupposes *mathematics as a metalanguage* for the study of reasoning. One gives a mathematical definition of what is meant by a “universe of discourse” (a “structure”) and by the “validity” (truth) of a statement in a universe. Then one can show (!) in the frame of the metalanguage that, for example, the above inference rule is correct. Namely, one can show that, in any universe in which not for all  $x$  ( $A$ ) is valid, also there exists  $x$  such that (not  $A$ ) is valid. This is an example for taking a crude tool, namely naive mathematical reasoning, for refining (another copy of) itself.

Here one should pause for a moment and consider the following fact that gives another argument why and how it is possible to use a tool for refining itself. For proving on the metalevel that, for example, the above inference is correct in all (infinitely many) possible instances of application it is necessary to apply certain inference rules, on the metalevel, only finitely many times. Each of these finitely many instances of application can also be viewed as an observation (insight, "intuition") in the particular situation. In short, by bringing together finitely many observations on the metalevel we can establish a rule that governs infinitely many instances on the object level.

Mathematics can now be characterized as the science that aims at establishing, by finite chains of reasoning, i.e. finitely many applications of (formal, controllable, and correct) inference rules, that a statement  $B$  follows from a statement  $A$ . Here, " $B$  follows from  $A$ " (or " $B$  is a semantical consequence of  $A$ ") is defined to mean that " $B$  is valid in all universes in which  $A$  is valid". Note that in general it is not possible to algorithmically determine whether a formula  $B$  follows from a formula  $A$  by just applying this definition of "follows" because, for doing so, one would have to check all possible universes and, in each universe, one would have to check the validity of  $A$  and  $B$ , which again may be a non-constructive process. The aim of mathematics can, therefore, also be rephrased by saying that it aims at establishing that a formula  $B$  follows from a formula  $A$  by finding a chain of reasoning (a "proof") consisting of individual steps governed by (algorithmically controllable) inference rules. Very concisely, *mathematics is the science that establishes semantical consequences by formal reasoning.*

#### 1.1.2.4 Reasoning Must Be General

In the preceding subsection we have already seen that inference rules are meant to be applicable in a wide range of situations. Whereas an observation refers to one particular situation and one observation is independent of the other, reasoning aims at being *general*, i.e. applicable in whole classes of situations. Actually, the evolution of logic has led to *universal* reasoning systems that can be applied in all possible universes and situations.

In fact, a rule of inference that would be applicable in only one instance is essentially an observation. Hence, generality is crucial for reasoning. Otherwise the purpose of reasoning, to establish truth without observation, could not be achieved.

*Note Bourbaki's 50 volumes describing a true  
the entire mathematics in a universal language (based on  
set theory).*

#### 1.1.2.5 Reasoning Versus Being

"Intellectual problem solving" as described above as a specific combination of observation, reasoning, and action is only one aspect of human existence. Intellectual problem solving should be considered as a *tool* that can be taken and also be put aside. When used in the right way, the use of this tool should not create bondage but freedom. The tool should never be our master.



The art of using the tool of intellectual problem solving without being deceived by its usage and bound by its effects is, by itself, not a theme that can be treated by intellectual means. Other layers of human consciousness must be invoked in order to set the frame for an appropriate use of the tool in freedom from bondage.

Action in freedom is the supreme ideal of "Eastern" consciousness-oriented culture, see for example the Bhagavad Gita, a part of the Vedas. It starts from the insight that a man must learn to sink back, any time, into the unified field at the basis of subject and object, real world and model, thinker and thought, knower and known in order to remain stable, free and evolutionary while acting. The technique of experiencing the unified field at the basis of subject and object is called "*meditation*".

The integration, in one's life, of the power of the "Western" intellectual technology of real world problem solving, and the "Eastern" meditative technology of being grounded in the unified field while acting, is the true challenge of our generation.

### 1.1.3 What is (Mathematical) Logic?

*Logic is the science of reasoning.* As in any science, knowledge in logic can be obtained by observation and by reasoning.

Mathematical logic uses the method of mathematics, i.e. reasoning, for studying mathematical reasoning. In the author's view, mathematics cannot be defined by the classes of objects studied but only by its method, namely reasoning. Therefore, there is no distinction between "mathematical reasoning" and "non-mathematical reasoning". Thus, in short, one could also say that *mathematical logic is reasoning about reasoning*.

There are other terms that have a meaning similar to "mathematical logic", namely "formal logic", "symbolic logic", and "metamathematics". In fact, these terms are used quite interchangeably. "Symbolic" and "formal" stress the fact that reasoning in abstract models is studied. However, we just explained above that this is the fundamental characteristic of reasoning. "Metamathematics" stresses the fact that mathematics is studied by mathematical methods. Historically, maybe "metamathematics" is slightly more restricted than "mathematical logic" because metamathematics sometimes refers to a certain school of thought in the foundation of mathematics that emphasizes constructive (algorithmic) methods for studying reasoning.

### 1.1.4 Is Reasoning About Reasoning Possible?

Reasoning about reasoning seems to be a contradiction in itself. How can a tool work on itself? There is nothing mysterious about that, however. Sharpening a tool by applying the tool is as old as mankind and, in fact, this self-referential process seems to be the very motor of intellectual evolution.

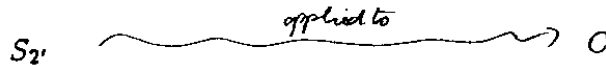
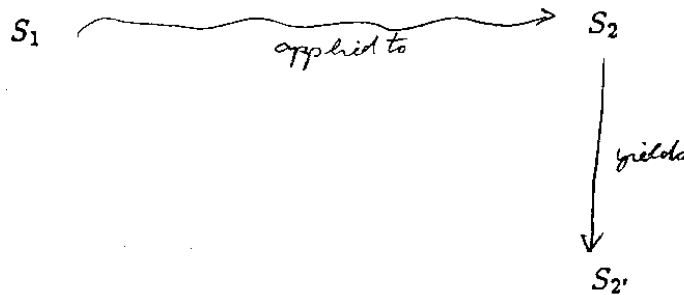
&lt;

He  
Su  
per  
to  
in  
it  
the  
with  
the  
an  
per  
it  
ed

**Example 1.1. (A Movie About an Early Ancestor at Work)**

Scene 1: There are two stones  $S_1$  and  $S_2$ . In no respect is  $S_1$  more "refined" than  $S_2$ . Our ancestor takes both stones and hits with  $S_1$  on top of  $S_2$ .  $S_2$  breaks into two parts. We observe a sharp edge at one of the two pieces,  $S_2'$ .

Scene 2: Our ancestor takes  $S_2'$  and applies it to a cane  $C$  for producing a cusp.



What can we learn from this little story?

First, we see that something that is crude ( $S_1$ ) when applied to something that is also crude ( $S_2$ ) may well produce something that is refined ( $S_2'$ ). In short, a crude tool may produce a refined working part. "Corollary": Crude reasoning may well produce refined reasoning. } .. And

Second, we observe that something that was process-ed in the first stage ( $S_2$ ) is process-ing in the subsequent stage, i.e. a working part may become a tool. "Corollary": Reasoning refined by reasoning should achieve better results in problem solving. } Object  
Subject  
Object □

We repeat the idea behind the above little story in a modern version.

**Example 1.2. (A Modern Version of the Movie)**

Scene 1: A compiler (which is a program)  $P_1$  works on a program  $P_2$  as input and produces a new version  $P_2'$  of it that is "refined" in some way (namely it can be executed by a machine).

Scene 2:  $P_2'$  can now be taken and applied to input data  $D$ .

Again, we see that an object from a certain class (a program) can be applied to an object of the same class producing something refined.

Also, what is a working part (input data for a compiler) in one phase may become a tool (a program) in the subsequent phase. □

Summarizing, we see that the transition of an object (the *working part*) from the stage where it is processed to the stage where it is itself processing (i.e. is a *tool*) is at the core of technological evolution. By iteration, this process has lead from the early beginning of human "engineering" to the amazing achievements of present "high-tech" and reflects the self-evolutionary power of the intellect.

Reasoning about reasoning is just one instance of this self-referential process and, in fact, the most fundamental and powerful one.

### 1.1.5 Language and Logic

#### 1.1.5.1 Paraphrasing Models and Reasoning in Terms of Language

*Language* in terms of written and spoken words and thoughts is the traditional carrier of intellectual models, i.e. the material from which intellectual models are constructed. Reasoning is, hence, operation on linguistic objects. With the evolution of science and, in particular, the advent of computers our conception of language must be drastically broadened and, in fact, any collection of building blocks from which models can be formed should be conceived as "language".

In a language-oriented terminology, the above picture of modeling real or abstract domains and reasoning in models can be paraphrased in the following way. A language consists of individual *expressions* (sentences, terms, statements etc.). The expressions are models (pictures) of *entities* (situations, processes etc.) in the *domains*, (structures, realities, universes of discourse) one is interested in. The interest, normally, stems from a problem that should be resolved in the given domain. The entities (situations, ...) described by the expressions of a language is called the "meaning" of the expressions.

The structure (external form) of the expressions of a language is called the "*syntax*" of the language. The meaning of the expressions of a language is called the "*semantics*" of the language. (One also uses the words "syntax" and "semantics" in a slightly different way: syntax is also the *science of external structure* and semantics is the *science of meaning*.)

In fact, normally, the expressions of a language are formed from elementary expressions by rules (syntactic rules). In some way, these rules reflect (often in a very abstract form) the way the objects (situations, ...) in the described domains are composed from simpler objects. The elementary objects, the construction rules and, hence, compound expressions of a language should be "finitary" (manageable, algorithmic, simple). This corresponds to the overall objective of models. Models should be manageable pictures of the world modelled. Otherwise there would be no point for using models.

*Reasoning*, then, is operating on the expressions of the language with the goal of arriving at new expressions with an "interesting" meaning, e.g. at expressions that describe the solution to a problem in the described domain. The reader is encouraged to paraphrase the fundamental properties of reasoning (abstractness, controllability, correctness, and generality) in linguistic terms.

### 1.1.5.2 Some Distinctions

We introduce a couple of distinctions that are important when speaking about languages.

**Descriptive and Algorithmic Languages** “*Descriptive*” languages describe objects, situations, relations between objects etc. in contrast to the “*algorithmic*” (“*imperative*”) languages that prevail in computer science. Imperative languages describe processes, activities, functions, commands, procedures. Descriptive languages are the languages of traditional mathematics and, in fact, logic in the traditional sense treats only descriptive languages. Predicate logic is the most prominent example of a descriptive language. In a broad sense, logic is concerned with syntax, semantics of and “reasoning” in arbitrary languages and there is a continuous transition between the broad and the restricted use of the term “logic”. In computer science, the theory of algorithmic languages (programming languages) is also called “theory of formal languages”. The latter term, however, for a long time was only restricted to the syntax of programming languages. There is also the term “computational logic”, which is used either in the sense of logic (theory) of syntax and semantics of algorithmic languages or in the sense of logic of automated reasoning or in both senses together. All this shows that people more and more are aware that, essentially, the treatment of (descriptive and imperative) languages and reasoning (computing) in these languages is an integrated body of knowledge that cannot be separated in independent parts without losing its power.

**Universal and Special Languages** Reasoning and computing aims at generality. Most of the prominent descriptive and algorithmic languages of mathematics and computer science can even be shown to be “*universal*” in a very precise sense. For example, first order predicate logic is “universal” in the practical sense that all of mathematics can be described in it and also in the theoretical sense that the reasoning system of this language is so powerful that anything that is “true” in a semantical sense can be derived by applying the reasoning rules.

Universal languages are in contrast to “*special*” languages as, for example, the CSG (“constructive solid geometry”) language that allows to describe certain classes of geometrical objects, or the “language of switching circuits” that describes certain hardware-constructs, or “propositional logic” that allows to formulate certain simple compound facts etc.

**Natural Languages and Formal Languages** “*Natural*” languages as, for example, English or German are learned by “training in context”. This means that the syntactically correct expressions of these languages and the meaning of these expressions are established by repeated use in the situations the expressions describe. This is in contrast to “*formal*” languages. Their syntax and semantics is “defined” by definitions formulated in a “metalanguage”, i.e. a language that describes the syntactical objects (expressions) of the language,

the objects that are described by the expressions of the language, the relation between the syntactical objects and the objects described (the semantics of the language), and the rules of reasoning.

Note that the property of being “formal” has nothing to do with the syntactical appearance of a language. Being “formal” does not necessarily mean that the language is written with many symbols. It could equally well be written in (a subset) of plain English or any other “natural” language. The point is that, in a formal language, we do not rely on learning the concepts of the language by experiments in contexts (with the danger of ambiguity by non-identical experience from non-identical experimenters) but on defining them using an already available (natural or formal) language.

Conversely, a “natural” language can be quite “symbolic” and, still, is learned by experience in semantical contexts. For example, the magma of every-day mathematical language, though sometimes overloaded with symbols, is normally learned as a “natural” language, i.e. by training in semantical context.

*This means that English has long been treated as a formal language (!) in the traditional way (before World War II).*

**Metalinguage and Object Language** In the definition of a formal language, the syntax, semantics, and reasoning system of the language is defined in an already available (natural or formal) language. In this context, the language being defined is called the “*object language*” and the language used for defining it is called the “*metalinguage*”.

Note that the terms “metalinguage” and “object language” are no absolute terms.<sup>4</sup> It depends on the situation whether a given language is an object language or a metalinguage. For example, set theory (in the language of first order predicate logic) is a metalinguage when used for defining, formally, first order predicate logic. It can be an object language when it is being defined as a special theory of formal first order predicate logic. This is confusing for beginners of logic and should be thoroughly understood. It can be easily made clear by looking at our little story about the stone that may be used for refining another stone.

### 1.1.6 The Notion of Model in Logic

We said that mathematics is operating in (linguistic) models of real (or abstract) domains.

In the technical terminology of mathematical logic, however, the term “model” is just used the other way around: A “model” is a domain about which all statements of a given set of statements are true.

Hence, we say in the context of mathematics that “some set of statements is a model of a real world domain” whereas in mathematical logic we say that “some domain is a model for a set of statements”. It is important to be aware of this conflicting use of the term “model” in order not to be confused by the discussions in the technical parts of this book.

<sup>4</sup> See my book *Mathematics* page 9.

### 1.1.7 Some Elementary Notions Connected with Reasoning

In this subsection we give an informal explication of the notions “*axiom*”, “*theorem*”, “*definition*”, “*theory*” and of two approaches to the notion of a “*formal system*”.

Reasoning can derive new statements from given ones. If the initial statements are true in a certain domain then the derived ones are also true in the same domain. *Axioms* are statements that are taken as initial statements in logical derivations. Mathematics does not ask about whether these statements are true or not. Rather, mathematics derives statements from axioms with the intention that, if somebody (for example, a physicist) finds a domain in which the axioms are true (i.e. a “*model*” for the axioms in the terminology of logic), then he already has an enormous body of derived statements at his disposal. The statements derived from axioms are called “*theorems*”. A set of axioms together with all derived statements is called a “*theory*”.

In fact, also abstract domains of mathematics may be “*models*” for axiom systems. For example, each “*group*” is a model for the axioms of “*group theory*”. The “*axiomatic method*” opens a very powerful means for economical thinking: Instead of investigating the truth of certain statements for each individual domain, certain statements can be proven correct for a whole class of domains that satisfy certain axioms.

A “*definition*” is an axiom that describes how a new notion is connected with notions already introduced. Definitions must possess certain formal properties. First, it must be clear how each statement involving the new notion can be replaced by another (probably more complicated) statement that does not involve the new notion. Second, definitions must have a syntactical structure that guarantees that by adding definitions to a non-contradictory axiom system the theory stays non-contradictory.

Roughly, there are two approaches to formulating systems of inference rules (“*logical calculi*”, “*formal systems*”): Systems of the “*Hilbert type*” and systems of the “*Gentzen type*”. Both approaches are equivalent. However, they have a different flavor.

In *Hilbert systems* an inference rule describes how a new statement can be derived from given statements of a certain form. For example,

$$\frac{\neg\forall x(A)}{\exists x(\neg A)}$$

is a typical inference rule in a Hilbert system. A *proof* is a sequence whose statements are either axioms or statements that are derived from earlier statements in the sequence by application of an inference rule. This notion is simple and is often used in a context where one is more interested in the metamathematical results but not so much in the application of logic to actual mathematical proving.

Practical proving is not arranged in the way Hilbert systems describe it. Rather, it follows the following scheme. We consider “*proof situations*”. A proof situation is characterized by a set of statements that are “*known*” in the particular situation and a set of statements that still have to be proven (“*goals*”).

An inference rule, then, describes how certain proof situations can be replaced by certain other proof situations. For example,

$$\frac{A, B \vdash C}{A \vdash (B \Rightarrow C)}$$

is a typical inference rule in a Gentzen system. Here,  $M \vdash N$  describes the proof situation where all statements in the set  $M$  are known and the statements in the set  $N$  have to be proven. A proof in a Gentzen system is a tree of proof situations where each proof situation at a node can be derived from the proof situations at the predecessor nodes by application of an inference rule.

## 1.2 Logic Motivated by Mathematics

### 1.2.1 The Refinement of Mathematical Reasoning

The *refinement of mathematical reasoning* is one of the two main motivations for studying logic. In fact, over the centuries, by the self-referential process of reasoning about reasoning, the technique of intellectual problem solving by reasoning has reached an incredible sophistication and power.

In history, humans started from a stage where observation and reasoning was almost indistinguishable and, hence, results obtained by reasoning were only slight generalizations of observed facts. We are now in a situation where reasoning is understood so clearly that big parts of reasoning can be fully automated on computers and for the remaining parts it is clear why they cannot be automated or why they can be automated only at an enormous expense of time and space.

In practical terms, it can be observed that the study of logic has an extremely purifying and improving effect on the intellectual potential of students of mathematics. Some people believe that, for students of mathematics, it is useless to study logic because "one either has logic built in" or there is no way to remedy this. They admit that the study of logic is interesting in itself as a "philosophical" discipline but they do not see any practical value in the course of a regular mathematics curriculum.

In some sense they are right because most probably the "brainware" that underlies reasoning cannot be changed very much (although even this can be doubted because it seems that in the brain much depends on — early — learning). Also, mathematical logic mostly is presented only as the metascience of mathematics but its results are never applied for improving the style and quality of mathematics. This is deplorable, however. For changing this situation, mathematicians should learn from computer scientists. By the "software crisis", computer scientists have learned in a most painful way that style and formal quality is not only a matter of aesthetics but rather a practical necessity. To computer scientists, this lesson was taught by the fact that programs must run on machines and the deficiencies of software become readily apparent by failure and bad performance. The machine is brusque in rejecting bad software.

By contrast, the paper on which mathematicians write their proofs is patient. It does not reject anything. Therefore bad technical quality can persist much longer and, in fact, unintelligibility is sometimes confused with ingenuity. There is no apparent "proof crisis" in contemporary mathematics or, at least, mathematicians would not admit any. (A "proof crisis" could be inferred from the fact that, on average, mathematical papers are only read 1 1/2 times each. The proof crisis in the last century was different in nature. see next subsection.) Personally, I believe that it is high time that, in the math curricula, logic guides practical proving similarly as "software science" does guide programming.

It is true that, when studying logic, it is difficult to treat the actual results of mathematical logic together with the practical implications and applications these results have for every-day proving in mathematics. However, in the view



of the author, it is absolutely urgent that the study of logic in math curricula is augmented by courses that aim at practical proof training. In fact, this training should happen as early as possible in the curricula. An attempt in this direction (albeit devoted to first semester computer science rather than math students) is (Buchberger, Lichtenberger 1980).

The integration of logic and proof training into the curricula seems to be overdue in particular in view of the advent of a general science of automated problem solving that could be called "compumathics" or "mathformatics" and is nothing else than the implementation of the original spirit of mathematics as the technique of problem solving in models in the presence of powerful machines that allow to quickly implement any model.

### 1.2.2 The Solution of Foundational Problems

Improvement of the practice of mathematical reasoning is one motivation for studying logic. The other one is the solution of contradictions and deficiencies in certain advanced reasoning techniques that became apparent in the last century.

In fact, by the dramatic advancement and achievements of mathematics the last century *did* experience a "proof crisis" similar to the recent "software crisis" in computer science. Some of the difficulties seemed to be insurmountable and attracted the most brilliant minds. The combined effort of these mathematicians has led to a body of knowledge in logic that is surely one of the most exciting achievements of mankind. The explosive effect, both in quality and quantity, of the foundational questions raised in the last hundred years can be felt.

We give just one example of a paradox that puzzled mathematicians in the last century.

**Example 1.3. (Russell's Antinomy)** It is well known that the length of the diagonal of a square whose edges have length 1 is not a rational number. Of course, it must be a goal of mathematics to provide a "model" for everything that occurs in nature. Thus, the domain of rational numbers must be expanded by a suitable construction. One possibility is to take certain pairs of sets of rational numbers, called "Dedekind cuts". The collection of these cuts is a domain (the domain of real numbers) that is strictly bigger than the domain of rational numbers and, in fact, contains also a number that faithfully describes the diagonal in the above square. (A "cut" is a pair  $(A, B)$  of sets of rational numbers such that for all  $x \in A, y \in B, x < y$ . Note that this notion makes it possible, for example, to speak about numbers less and greater than the number that should measure the above diagonal without actually mentioning this number.)

In the exact definition of a cut, one needs the notion of a "set". This notion goes beyond the notion of a number. Intuitively this new notion has one characteristic property:

For all "properties"  $E$  there exists the "set" of all objects  $x$  that possess property  $E$ .

Hence, it is near at hand that the theory of sets should contain the following axiom for every property  $E$  with a free variable  $x$  that can be formulated in set theory:

there exists  $M$  such that for all  $x$  ( $x \in M$  iff  $E$ ),

where  $M$  is a variable that does not occur "free" in  $E$ .

Assumption of these axioms, however, readily leads to the following contradiction. We consider the following property

$x \notin x$ .

Applying the corresponding axiom we obtain

there exists  $M$  such that for all  $x$  ( $x \in M$  iff  $x \notin x$ ).

For this  $M$  we have

for all  $x$  ( $x \in M$  iff  $x \notin x$ ).

In particular, for  $x := M$  we obtain the contradiction

$M \in M$  iff  $M \notin M$ .

Thus, we have a situation where we have added something to our mathematical knowledge that seems totally plausible but leads to a contradiction. In fact, this particular contradiction is removed in modern set theory by restricting the above set of axioms in the following way. For every property (i. e. formula of set theory)  $E$  with free variable  $x$  and every variable  $B$  not occurring free in  $E$  one stipulates<sup>①</sup>

*(compare axioms 5.4 in chapter 5) Basic set (Grundmenge)*

for all  $B$  there exist  $M$  such that for all  $x$   
 $x \in M$  iff ( $E$  and  $x \in B$ ).

However, how can we be sure that from this restricted set of axioms no other contradiction can be inferred? This was one of the questions that stimulated modern research in mathematical logic. In the case of this particular question, the answer is negative in the sense that one can show that a proof that establishes that a given axiom system for set theory, on the metalevel, needs more powerful methods than are available on the object level. Still, work on this question has not only led to exciting insights into the technique of reasoning but, interestingly enough, long before the advent of the first computer the analysis of this question has produced an exact notion of what should be called "algorithmically computable".  $\square$

Here are some examples of basic problems in the foundation of mathematics that arose in the last century and have led to the present body of knowledge in mathematical logic:

*= consistency property of an axiom system*

- (ia) The Problem of Freedom from Contradiction, Given an axiom system for an area of mathematics. Question: Can reasoning ever deduce a contradiction from these axioms?

<sup>①</sup> In modern set theory the variable  $B$  is restricted to be a set. This is not the case in the original formulation of the axiom.

→ so there are procedures

25 **Mechanizability of Reasoning** Can the activity of reasoning be decomposed into so small and simple pieces that each individual step can be realized by a machine (a computer)?

A positive answer to this question was an early by-product of the fundamental analyses in modern mathematical logic.

→ properties of a proof system

26 **Completeness of Reasoning** Starting in the time of the ancient Greeks rules of inference were formulated more or less explicitly. Some of these rules were quite universal as, for example, the rules of propositional logic (covering the use of the connectives "not", "and", "or" etc.) and the rules of syllogistics (covering the use of the construct "all ... are", "some ... are"). Other reasoning techniques were only applicable in certain domains as, for example, induction in the domain of natural numbers.

Adding more and more techniques of reasoning expands the domain of theorems that can be inferred from basic facts. Given a particular system of inference rules one may ask: Can we add correct inference rules and thereby expand the set of derivable sentences? Or is the system "complete"?

It was one of the significant achievements of this century to establish that, in fact, for the important system of first order predicate logic there exists a complete set of inference rules (Completeness Theorem by Gödel, 1930).

→ property of an axiom system

↳ Also formulations say "consistent theory" (set of theorems has a model (i.e. a reality that covers))

27 **Completeness of Axiom Systems** For various fundamental mathematical domains one can try to establish systems of axioms (basic facts) from which, hopefully, all other sentences that are true in the domain can be inferred by reasoning. Question: Is a given axiom system powerful enough ("complete") for establishing all true sentences in the domain? Can we construct a "simple" complete axiom system? Here, "simple" could mean, for example, that at least it must be "decidable" whether or not a given sentence is an axiom. <sup>where?</sup>

It was another of the fundamental results by Gödel (1931) that, for example, for the domain of natural numbers the construction of a "simple" axiom system is impossible. <sup>So-called "incompleteness theorem" (bad name).</sup>

→ so there are no algorithms

36 **Decidability of Truth** Can the work of mathematicians be fully automated in the sense that one can construct an algorithm that decides, in finite time, whether or not a given sentence follows from given axioms. Or more specifically, can such an algorithm be constructed at least for certain areas of mathematics as, for example, the theory of real numbers under some simple operations like addition and multiplication.

Again, it was one of the dramatic advances in mathematical logic in the thirties to show that there cannot exist an algorithm that decides arbitrary sentences of first order predicate logic (Undecidability Theorem by Church, 1936). There are a number of deep recent results that show that, by contrast, there exist (complex) algorithms for deciding the truth of sentences in various extremely interesting restricted areas of mathematics (for example, Collins' algorithm for the theory of real closed fields, 1973).

*properties of an axiom system*

② **Categoricity of Axiom Systems** In the case, where it is possible to derive all sentences that are valid in a certain domain from a set of axioms it is interesting to know whether, by the set of axioms, the domain is completely characterized. Stated differently, given a set of axioms, is it possible to construct two or more "essentially different" (non isomorphic) domains that satisfy all axioms?

Amazing results have been achieved in pursuing this question. For example, domains can be constructed that satisfy all axioms of the well known Peano axiomatization of natural numbers but still are not isomorphic to the "ordinary" domain of natural numbers.

## 1.3 Logic Motivated by Computer Science

### 1.3.1 The Automation of Problem Solving

In Subsection 1.1.1 we have sketched the three steps of intellectual problem solving. Problem solving can be automated by delegating part of observation, reasoning and action to machines.

*Computer science aims at automating reasoning.* In its simplest form, operations in numerical models ("computations") are automated. This was and is the traditional scope of using computers. Probably, it is one of the most profound practical insights of this century that any systematic intellectual activity (reasoning) can be viewed as a kind of "computation" and hence, in principle, can be automated by using a computer.

The "history" of computer science can be characterized by increasing levels of automation in the transition from problem specification to solutions ("algorithms", "programs") in the world of models.

For exemplifying this view, we draw a slightly more detailed picture of the transition from problem specification to program:

Problem Specification

⇓ Proving

Mathematical Knowledge that is Useful  
for an Algorithmic Solution of the Problem

⇓ Programming

Program for Abstract Machine

⇓ Compilation

Program for Runtime Machine

⇓ Interpretation (Run-Time System)

Program for Hardware Machine

According to the four steps in the above picture, there are four possibilities for automating problem solving. On the one hand, one can try to automate proving and programming for bridging the gap between problem and program. On the other hand, one can try to establish higher and higher abstract machines by having more sophisticated compilers and run-time interpreters. Doing so, the gap between problem and program is not bridged but made smaller.

All these possibilities for automating problem solving have been amply pursued in computer science and the general tendency is towards higher sophistication. We sketch some land-marks in this evolutionary process.

### 1.3.2 Increasing Automation in the History of Computer Science

#### 1.3.2.1 Raising the Level of the Abstract Machine

Programming Style	Capabilities of the Abstract Machine
programming on the hardware level	operations on 0, 1-sequences, stored program
assembler programming	symbolic addresses
procedural languages	nested expressions, blocks, procedures
recursion	procedure stacks
dynamic data structures	garbage collection
functional programming	program = data
abstract data programming	general unification, direct implementations of term algebras
object oriented programming	generic dispatch
relational data base programming	predicate logic operations on finite sets
expert system shells	propositional theorem proving
logic programming	resolution theorem proving
constraint logic programming	special unification

#### 1.3.2.2 Bridging the Gap Between Problem and Program

Transition Paradigm	Techniques
program synthesis	extraction of algorithms from proofs, inference rules
program transformation	inference rules
program verification	verification condition generation, simplification of logical expressions, automated proofs of verification conditions

### 1.3.3 The Central Role of Logic for the Automation of Problem Solving

From the techniques listed in the preceding two subsections one sees that the direction computer science takes is towards more sophistication in the automation of problem solving by the use of more and more sophisticated techniques from mathematical logic.

Therefore, mathematical logic becomes more and more the mathematical basis for computer science. This is reflected even in some recent computer science curricula, see (Buchberger, Lichtenberger 1980) for the first radical approach in this direction. The role of logic for computer science was also concisely summarized in the recent book (Manna, Waldinger 1985): "Logic plays a fundamental role in computer science similar to that played by calculus

in physics and traditional engineering. A knowledge of logic is becoming a practical necessity for the computer professional."

### 1.3.4 Logic, Automated Reasoning and Symbolic Computation

Symbolic computation is the branch of mathematics and computer science that studies algorithms on symbolic objects, i.e. "finitary" (computer-representable, "simple") objects that represent (are "symbols", symbolic representations for) infinite or, at least, "complicated" objects. Problems in the abstract mathematical domains of non-finitary objects are (partially) reflected by problems in the finitary representation domains and the algorithmic solution of these problems is the scope of symbolic computation.

Expressions in the language of mathematics are, of course, symbolic objects. They are "finitary" but represent non-finitary situations in non-finitary domains. They have "meaning". Operation on these expressions is reasoning. We have seen that mathematical logic is the basis for the automation (algorithmization) of reasoning ("automated reasoning", "automated theorem proving", "computer-aided demonstration"). Automated reasoning is, hence, a branch of symbolic computation and, in fact, an important and basic one. And mathematical logic is, hence, one of the important foundations of symbolic computation.

## 1.4 A Synopsis

We have seen that the study of mathematical logic is motivated by the desire for improving the quality of mathematical reasoning and the necessity to overcome apparent difficulties that arise in a sloppy application of intuitively correct reasoning techniques. Furthermore, for computer science, mathematical logic is the fundamental tool that leads to progressive automation of problem solving.

Essentially, all these sources of motivation are just one: Reasoning is the fundamental basis of intellectual problem solving. Mathematical logic improves this tool and thereby improves human intellectual problem solving, makes it universally applicable and, finally, establishes the basis and explores the inherent limits for the complete automation of problem solving.

We have seen that mathematical reasoning is the essential ingredient in the intellectual problem solving technique on which our Western civilization is based. Mathematical logic is the instrument for purifying mathematical reasoning. Thus, studying mathematical logic is like polishing a mirror that reflects the accumulated and concentrated problem solving experience of our civilization.

We have also seen that intellectual problem solving based on logic is not an aim in itself but a tool that must be embedded into and governed by a wholistic experience of the unified field that lies at the basis of and goes beyond intellectual reasoning.

See

## 2. The Language of Predicate Logic

### 2.1 The Importance of Predicate Logic

In this chapter we define the language of predicate logic, in particular first order predicate logic. This language is of central importance for human problem solving because

1. it is general enough to be a formal frame for all of mathematics,
2. it can serve as a formal frame for computer science, and
3. it is the language on which, during the past decades, most of the foundational research effort concentrated, and which, probably for a long time, will be a primary model for the rigorous study of formal languages both in mathematics and computer science.

#### 2.1.1 Predicate Logic as a Frame for Mathematics

Predicate logic, as a means for expressing facts, is so universal that all results of mathematics can be conveniently described and proved in it. *The existence of such a language is by no means trivial.* In fact, it needed the effort of the most brilliant minds over more than two thousand years to expand the first tentative approaches to logic into the full universality of predicate logic. In particular, neither propositional logic nor Aristotelian logic is powerful enough to express and prove all facts of modern mathematics. The full expansion of predicate logic and its foundational analysis showing fundamental properties like "completeness" (Kurt Gödel 1930) and "undecidability" (Alonzo Church 1936) was achieved only in this century after a crucial contribution by Gottlob Frege in the last century (1879, "Begriffsschrift", the first complete syntactical presentation of predicate logic).

Predicate logic is a universal frame for mathematics even in the restrictive form of "first order predicate logic". First order predicate logic does not allow function and predicate variables but only function and predicate constants. In particular, no quantifiers ranging over function and predicate variables are allowed.

At first sight, in mathematics, function and predicate variables and quantifiers over function and predicate variables seem to be unavoidable. For example, in the definition

f is continuous at point x iff  
 for all  $\epsilon$  there exists a  $\delta$  such that  
 for all y with  $|y - x| < \delta$  we have  $|f(y) - f(x)| < \epsilon$

"f" is a function variable. In the sentence



for all  $f$   
 if  $f$  is differentiable then  $f$  is continuous

the function variable "f" is quantified by the "for all" quantifier. Similarly, in the definition

$p$  is bounded iff  
 there exists a  $B$  such that  
 for all  $x$  with  $p(x)$  we have  $|x| \leq B$

"p" is a predicate variable. In the sentence

for all  $p$   
 if  $p$  is bounded then  $p$  possesses an upper limit

the predicate variable "p" is quantified, again, by the "for all" quantifier. In these and other typical examples, *it seems that we need higher order predicate logic* (i.e. predicate logic with various layers of variables and corresponding layers of quantifiers) in order to express these more complex mathematical facts.

This apparent difficulty in using first order predicate logic as a universal language for mathematics can be resolved by *formulating, in first order predicate logic, the theory of "sets"*. In this theory, one introduces, first, a binary function constant "application" (let us write " $f \in x$ " for "f applied to x") and a binary predicate constant "is element of" (one writes " $x \in S$ " for "x is element of S"). Second, one formulates a number (in fact, an infinite number) of axioms that describe the basic properties of " $\in$ " and " $\in$ ". (Usually, only " $\in$ " is introduced as a basic concept and " $\in$ " is then reduced to " $\in$ " by a definition.) All variables in this theory range over *one type of "objects" ("sets")*. Functions and predicates are now just special sets, namely "relations" (i.e. sets of "tuples") and no extra variables for functions and predicates nor quantifiers over function and predicate symbols are needed.

Everything that can be expressed in higher order predicate logic can now be expressed in the special first order predicate logic language involving the "non-logical" constants " $\in$ " and " $\in$ ", i.e. in "first order set theory" (or, more specifically, in "Zermelo-Fraenkel set theory").<sup>①</sup> For example, the above definitions become

$f$  is continuous at point  $x$  iff  
 for all  $\epsilon$  there exists a  $\delta$  such that  
 for all  $y$  with  $|y - x| < \delta$  we have  $|f(y) - f(x)| < \epsilon$

and

$p$  is bounded iff  
 there exists a  $B$  such that  
 for all  $x$  with  $x \in p$  we have  $|x| \leq B$ .

<sup>①</sup> For a detailed treatment of this theory, see, for example, ...

In these definitions, the variables " $f$ ,  $x$ ,  $\epsilon$ ,  $\delta$ ,  $y$ ,  $p$ ", and " $B$ " are all of the same, namely first, "order", none of them is of higher order. They all range over the same universe of "objects" or "sets". " $\cdot$ " and " $\in$ " are constants and are never quantified! " $f$ " and " $p$ ", however, are quantified for example in

for all  $f$   
if  $f$  is differentiable then  $f$  is continuous

and in

for all  $p$   
if  $p$  is bounded then  $p$  has an upper limit.

However, now, this quantification does not lead outside first order predicate logic because " $f$ " and " $p$ " are of the same order as all the other variables. (For a practical understanding and training in the subtle difference between having function (predicate) variables and using " $\odot$ " (" $\in$ ") together with first order variables, see (Buchberger, Lichtenberger 1981, pp. 112)).

Thus, from a practical point of view, first order predicate logic plus the axioms of set theory and higher order predicate logic have the same expressive power. (When making subtle theoretical distinctions that will be possible only much later in this book, higher order predicate logic is strictly more powerful than first order set theory.) However, formulating all of mathematics in first order predicate logic has advantages because the language and its proof mechanisms are simpler. Therefore, I think that staying within first order predicate logic by using *set theory as a "universal language environment"* is a good choice for practical mathematics because it is powerful and well understood at the same time. In fact, the cooperative work of the "Bourbaki" group of mathematicians in the past few decades shows that it is conveniently possible to formulate the existing body of mathematical knowledge in the uniform frame of first order set theory.

### 2.1.2 Predicate Logic as a Frame for Algorithmic Problem Solving

As we have seen, (first order) predicate logic is a universal frame for expressing and proving mathematical concepts and facts. It is therefore also a universal frame for *specifying problems*.

Recently, based on earlier work on the completeness of first order inference mechanisms, it has been shown how proving in first order predicate logic can be automated (John Alan Robinson 1965 and subsequent research) and that, in fact, *proving* the existence of solutions to problems for concrete input parameters is *equivalent to computing*.

First order predicate logic can therefore provide a uniform frame for rigorous problem specification and algorithmic problem solving. The feasibility of the predicate logic approach as a common basis for problem specification and solution has been impressively demonstrated by the success of the *logic programming* (PROLOG) approach to programming.

Furthermore, predicate logic has equally well shown its unifying potential and practical power in the area of data design and analysis (*abstract data types, relational databases*). Structuring problem specifications, structuring algorithms and structuring data are three orthogonal but complementary approaches to algorithmic problem solving. *Predicate logic is a common frame for all three of them.*

Still, it may well be possible in the future that formal languages based on fundamental concepts other than the concept of “set” will turn out to be more appropriate. For example, when speaking about “algorithmic domains” in the area of generic algorithms and data types the set theoretic notion of a function is too coarse for capturing the idea that two domains with identical operations are distinct when the operations are realized by different algorithms. As an alternative to predicate logic and set theory, extended versions of the  $\lambda$ -calculus promise to give a more natural frame for the evolving integration of “static” (“descriptive”) and “dynamic” (“algorithmic”) mathematics or, oversimplified, of mathematics and computer science. In fact,  $\lambda$ -calculus can be viewed as a theory whose basic concept is “ $\odot$ ”, application, whereas set theory is the theory whose basic concept is “ $\in$ ”, membership.

However, I guess it will still take some time until alternatives to predicate logic will be mature enough to replace it. Also, it is too early to make a prediction whether a complete replacement is really desirable and possible. Because of its practical importance, power, and maturity, we will make first order predicate logic the main topic in this book.

### 2.1.3 Predicate Logic as a Model for Formal Language Studies

Predicate logic evolved from informal language constructs describing mathematical facts. These constructs were purified and tried out in the demanding environment of mathematical argumentation by generations of mathematicians. Finally, in an enormous effort since the middle of the 19th century, the syntax, semantics, and proof system of predicate logic was rigorously defined and the relation between syntax, semantics and proving was thoroughly investigated. Some of the most, if not the most, exciting insights of all human history were gained by these investigations.

Many fundamental problems, concepts, approaches, and intellectual techniques evolved, for the first time in history, in the foundational research on predicate logic: The subtle distinction between language layers (the “meta” and the “object” layer), the careful distinction between syntactical and semantical concepts, the fundamental interplay between semantics and formal manipulation on language constructs (proving), the clear formulation of problems relating syntax, semantics and proving (for example, “completeness”, “categoricity”, “definability”, “decidability”), the technique of inductive definition of syntax and semantics, the technique of constructing semantical models from syntactical material, the technique of bootstrapping powerful languages from small kernels and many, more refined, techniques. These *achievements of predicate logic research* provide now a sound and sophisticated *technology for the design*

and analysis of many other formal languages, in particular, in computer science. In a slight oversimplification, computer science may also be characterized as "automation of problem solving by language design and implementation". The role of predicate logic as a basis for computer science can, therefore, not be overestimated.

## 2.2 Characteristics of (First Order) Predicate Logic

Predicate logic is a *descriptive language*. The main constructs ("formulae", "propositions", "statements") of the language of predicate logic describe facts about objects in various domains of discourse. Auxiliary constructs (the "terms") describe objects. Predicate logic is, hence, essentially a *"descriptive" language*. However, the logic programming approach teaches that, in fact, predicate logic can also be viewed as an algorithmic language. (In this approach, problem specifications are "programs" that initiate proofs. The proofs are the "computations" in the corresponding abstract machine, which essentially is a first order theorem prover.)

Predicate logic is a *universal language*. As explained in Section 2.1.1, predicate logic is suitable for any domain of discourse.

Predicate logic, as treated in this book, is a *formal language*, i.e. its syntax and semantics is defined in a *metalanguage* whose availability is presupposed.

It turns out that, as a metalanguage for defining (first order) predicate logic, we need some formulation of set theory. In this book, we use Zermelo-Fraenkel set theory (viewed as a natural or a formal language) as *the metalanguage for the study of predicate logic*, i.e. we use a special first order theory as a metalanguage for the study of first order predicate logic. For keeping the metalevel and the object level apart, we use the following notational convention in this book: All symbols of the object language are written in typewriter notation. For example, "r" is a symbol of the object language whereas "f" and "f" are symbols of the metalanguage.

## 2.3 Informal Syntax and Semantics of First Order Predicate Logic

The language of predicate logic has *"variables"* ("individual variables", "object variables") like "x", "y", ... that range over arbitrary elements in a "universe of discourse", for example the set of natural numbers or the set of screws in a box etc.

Furthermore, there are *"function symbols"* ("function names", "function constants") that can denote functions (operations, processes) on the universe of discourse. For example, "+" and "." are frequently used function symbols that, most times, denote addition and multiplication. However, in other universes of discourse they may well denote other operations, for example, "+" may stand for "union". Some of the function symbols may denote "0-ary" functions, i.e. functions with no arguments. These function symbols are often

called "constants" or, more specifically, "individual constants". The 0-ary functions denoted by constants produce exactly one value. Hence, they are in 1-1 correspondence with the elements in the universe of discourse and are often identified with these elements.

Similarly, there are "relation symbols" ("predicate symbols") that denote relations (predicates, attributes, properties) between (of) elements in the universe of discourse. For example, " $<$ " and " $|$ " are often used as predicate symbols and, usually, denote the "less-than" and the "is-divisible" relation, respectively.

In general, we are totally free in choosing function and predicate symbols and in attributing functions and relations as "denotation" ("interpretation", "meaning", "semantics") to these symbols.

From the variables and the function symbols, one can build more complicated language constructs, called "terms". The "nesting" of terms can be arbitrarily deep. For example, " $x$ ", " $x+y$ ", " $x \cdot 2 + (x - y) \cdot (x)$ " are nested terms. A term does not denote any particular element in the universe of discourse. Only after assigning "values" (objects in the universe of discourse) to the variables occurring in a term, the term denotes a particular object. For example, after assigning 2 to " $x$ " and  $-2$  to " $y$ ", " $x + y$ " denotes 0.

Terms are only auxiliary constructs in first order predicate logic. The main constructs are the "formulae" ("propositions", "assertions", "sentences", "statements").

The simplest type of formulae are the "atomic formulae". An atomic formula consists of a predicate symbol and several terms. For example " $n \cdot n > 2 \cdot n$ " is an atomic formula. An atomic formula asserts that the relation denoted by the predicate symbol holds for the objects denoted by the terms. Such an assertion can be true or false. In general, the truth or falsity will depend on the objects denoted by the terms in the atomic formula. As explained above, these objects depend on the assignment of values to the variables occurring in terms.

Many *different notations* are in use for showing the nested structure of terms and atomic formulae. For example, binary function and predicate symbols are mostly written "infix" (e.g. " $x < y$ "), others are written "prefix" (e.g. " $\text{GCD}(m, n)$ "), unary function symbols are sometimes written "postfix" (e.g. " $n!$ ").

Given formulae  $A$  and  $B$ , more complicated formulae can be built up by connecting formulae with the "propositional connectives" "not", "and", "or", "if ... then", and "if and only if". Again, many different notations are in use for these connectives. For example, " $\neg$ ", " $\implies$ ", " $\supset$ ", "implies", "from ... follows" etc. are some of the notations in use for the "if ... then" connective. Formulae of the forms "not  $A$ ", " $A$  and  $B$ ", " $A$  or  $B$ ", "if  $A$  then  $B$ ", " $A$  if and only if  $B$ " are called "negations", "conjunctions", "disjunctions", "implications" and "equivalences", respectively. In a given universe of discourse and under a given assignment of objects to variables, " $A$  and  $B$ " is true iff both  $A$  is true and  $B$  is true. " $A$  or  $B$ " is true iff at least one of the two,  $A$  or  $B$ , is true. "if  $A$  then  $B$ " is true iff at least one of the two, "not  $A$ " or  $B$  is true. The latter stipulation may seem to be "unnatural". Consider, however, the case of a formula of the form "for all  $x$  (if  $A$  then  $B$ )"

where  $A$  and  $B$  may contain the variable  $x$ . In this case, depending on the assignment of objects to the variable  $x$ ,  $A$  may become true or false. "for all  $x$  (if  $A$  then  $B$ )" is true iff, for any assignment,  $A$  becomes false (in this case we do not care about  $B$ ) or  $A$  becomes true, in which case  $B$  must be true also. This is equivalent to saying "not  $A$ " is true or  $B$  is true. " $A$  if and only if  $B$ " is true iff both,  $A$  and  $B$ , are true or both are false.

Finally, given a variable  $x$  and a formula  $A$ , one can form the formulae "for all  $x$   $A$ " and "there exists an  $x$  such that  $A$ ". Such formulae are called "universal formulae" and "existential formulae", respectively. "for all" and "there exists ... such that" are called the "universal quantifier" and the "existential quantifier", respectively. The variable  $x$  becomes "bound" by the universal or existential quantifier. "for all  $x$   $A$ " is true iff, for all assignments of objects to the variable  $x$ ,  $A$  becomes true. "there exists an  $x$  such that  $A$ " is true iff there exists an assignment of an object to the variable  $x$  such that  $A$  becomes true.

Here is an example of a predicate logic formula in one of the many notations in use:

$$\forall \epsilon \exists \delta \forall y (|y-x| < \delta \implies |f \odot y - f \odot x| < \epsilon).$$

The variables " $x$ " and " $f$ " are "free" (not bound) in this formula. All the other variables are bound by quantifiers.

In a formal treatment of first order predicate logic, one must unambiguously define a specific syntax for the formulae. In this book, we will choose a particular syntax that allows easy typing of the formulae on a keyboard and, in fact, is very close to natural language syntax. However, the particular choice of a particular syntax has no theoretical importance. The really important information contained in the syntax is the specification of the distinct "types" of linguistic objects contained in the language, the specification of "constructors" by which more complex linguistic objects can be built from elementary ones, and the specification of "selectors" by which complex linguistic objects can be decomposed into more elementary ones.

Therefore, in modern approaches to syntax, instead of giving one concrete syntax axioms are formulated that describe the properties of the types, constructors, and selectors of the language ("abstract syntax").<sup>6</sup> This has the advantage that assertions about the syntax of the language are valid for all possible realizations of the language, e. g. in a computer, and are completely independent of notation, data structures etc. In order not to confuse the reader by too much abstraction, we prefer to pursue the concrete syntax approach in this chapter. However, the definitions will be structured in such a way that the essential structural information that would form the body of the abstract syntax can be extracted easily from the concrete syntax.

For putting first order predicate logic to practical work in mathematics and computer science, it is absolutely necessary to acquire *fluency in formulating facts and proving theorems in the language of predicate logic*. Except for some simple exercises, we cannot spend much space in this book on achieving this important goal. The reader who does not feel at ease with the practical use of

predicate logic as a language is referred to (Buchberger, Lichtenberger 1980), where ample space is devoted to training the use of predicate logic in very many different syntactical representations including those that, in their appearance, do not differ very much from plain English. The main part of (Buchberger, Lichtenberger 1980), then, treats the technique and art of practical proving.

Let us, finally, make a remark about the usage of the expression "language of (first order) predicate logic". When we say "*the* language of first order predicate logic" we mean the language that is built up from variables and arbitrary (or "all conceivable") function and predicate symbols by forming terms, atomic formulae, propositional formulae and quantifier formulae. When we say "*a* first order predicate language" we mean a particular instance of "the language of predicate logic" that is characterized by the particular (few) function and predicate symbols needed to speak about the given universe of discourse. For example, for speaking about natural numbers, in some situations we may confine ourselves to using only "+", "." and "<". In other situations we may wish to include also the symbols "!" and "|" in our language. These are different "languages of first order predicate logic". Thus, "*a* language" of predicate logic is determined by its function and predicate symbols (the "non-logical symbols") whereas "*the* language" of predicate logic is the general linguistic frame for all these specific languages.





## 2.4 A Particular Formal Syntax for First Order Predicate Logic

In this section, we will define one particular syntactical representation of first order formulae. In this syntax, formulae are special "strings" (or "expressions"), i. e. sequences of "symbols". We will need some symbols as "variables", some symbols as "function constants" and "relation constants", some other symbols as "connectives" and "quantifiers", and finally some auxiliary symbols like "(", ")" and ", ". A sequence consisting of the symbols  $s_1, s_2, \dots, s_n$  is written in the form  $s_1 s_2 \dots s_n$  with spaces between the individual symbols. (Formally,  $\sigma$  is a string over a set  $S$  of symbols iff  $\sigma: \{1, \dots, n\} \rightarrow S$  for some  $n \in \mathbb{N}$ . Thus, the notation  $s_1 s_2 \dots s_n$  is only an abbreviation for describing a particular function.) Some of the "symbols" used in our syntactical representation of predicate logic will actually be English words like "sinus" or even groups of words like "for all". We consider such words and groups of words as atomic symbols whose possible decomposition into individual letters is irrelevant. If there is danger of ambiguity we sometimes will write 'for all' instead of for all etc. in order to emphasize that 'for all' is to be considered one symbol. Thereby we will be able to distinguish between, for example, not less( $x, y$ ), where not is a connective and less is a binary relation constant, and 'not less'( $x, y$ ), where 'not less' is a binary relation constant. However, most times, we omit the quotation marks because we can safely trust that the "reader" (which can be a machine) is able to parse the strings correctly (in the "scanning" phase of analyzing the string).

In fact, any object can be used as a symbol and we will need this view in some of the later investigations. In the examples of concrete formulae, however, we will encounter only the usual symbols of mathematics that are composed of the letters and digits of a few alphabets.

The version of predicate logic defined in the first subsection will be called "kernel language". It is the actual object of study in this book. The number of language constructs of the kernel language will be kept small in order to make *studying* the language easier.

For practical purposes, i. e. for *using* the language, the kernel language is not suitable because the formulae of the kernel language tend to be complex and hard to read<sup>1</sup>. Therefore we extend the language in a number of ways in the subsequent subsection. However, the formulae of the extended language are only viewed as abbreviations of the longer, and less readable, formulae of the kernel language. With each type of formulae of the extended language we will specify the abbreviated formulae of the kernel language. The formulae of the extended language are not the object of study of the book. The extended language is mainly used in the examples for discussing concrete formulae.

<sup>1</sup> So object language should be kept small for sake of readability but not too small should be a good idea to make it be "compact".

<sup>2</sup> Of the *not less* phrase, it is not clear if only a formal of *not less* is intended, but it is not clear.

### 2.4.1 The Kernel Language

The symbols not, or, and 'for some' (and some other symbols of the extended language) are called "special symbols". They will not be used as variables or constants.

**Definition 2.1. (Variables)**  $V$  is a set of variables iff  $V$  is a denumerable infinite set of words consisting of an English letter and subsequent letters and or digits and disjoint from the set of special symbols.

**Example 2.2. (Variables)** The following symbols may be elements in a set of variables:  $x, y, F, F', \text{epsilon}, \text{eps1}$ .

Let now some set  $V$  of variables be fixed.

**Definition 2.3. (Constants)** *Do not confuse with "domain" (Def 2.20)!*

$(F, R, A)$  is a domain of constants iff

- $F$  is a set of symbols ("the set of function constants"),
- $R$  is a set of symbols ("the set of relation constants"),
- $A: (F \cup R) \rightarrow \mathbb{N}_+$  ("the arity function"),
- $\in \in R, A(\in) = 2,$
- $F$  and  $R$  are disjoint,
- $F$  and  $R$  are disjoint from  $V$  and the set of special symbols. □

In the above definition we require that the binary equality symbol "=" is always among the relation constants. This variant of the predicate logic is called "predicate logic with equality". The equality symbol will be treated in a special way both in the definition of semantics and in the inference rules. It would also be possible to study "predicate logic without equality". However, most of the mathematical theories are more naturally described in predicate logic with equality. On the other hand, logic programming in its elementary form, is based on predicate logic without equality. The inclusion of "=" as a special concept into predicate logic has advantages and disadvantages. We decide to present the variant with equality in this book.

Let now  $(F, R, A)$  be a domain of constants. *i.e. of first-order predicate logic*

**Definition 2.4. (First Order Terms)** The set of (first order) terms (over  $(F, R, A)$ ) is the smallest set of strings that satisfies the following conditions:

If  $v \in V$  then  $v$  is a term.

If  $f \in F, A(f) = n,$  and  $t_1, \dots, t_n$  are terms then  $f t_1 \dots t_n$  is a term. □

The specific syntax we are using here is called "Polish notation" or "prefix notation". It avoids the use of parentheses. For theoretical purposes prefix notation is nice because it concentrates on the essential structural information and omits all "syntactic sugar". It is the type of concrete syntax that is closest to "abstract syntax".

*actually, completely true*

*"or" is not a special symbol, it is a logical connective*

**Definition 2.5. (Atomic Formulae)** The set of (first order) *atomic formulae* (over  $(F, R, A)$ ) is the smallest set of strings that satisfies the following conditions

If  $r \in R$ ,  $A(r) = n$ , and  $t_1, \dots, t_n$  are terms  
then  $r t_1 \dots t_n$  is an atomic formula.

**Definition 2.6. (First Order Formulae)** The set of (first order) *formulae* (propositions, or statements (over  $(F, R, A)$ )) is the smallest set of strings that satisfies the following conditions:

If  $a$  is an atomic formula then  $a$  is a formula.

If  $p$  is a formula then  $\text{not } p$  is a formula.

If  $p_1$  and  $p_2$  are formulae then  $\text{or } p_1 p_2$  is a formula.

If  $v$  is a variable and  $p$  is a formula then  $\text{for some } v p$  is a formula.

**Example 2.7. (Terms and Formulae)** Let  $V$  contain the symbols  $x, y, f23, f$ ,  
g. Let  $F := \{+, !\}$ ,  $R := \{<, \text{'is bounded'}\}$ ,  $A(+):= 2$ ,  $A(!):= 1$ ,  $A(<):= 2$ ,  
 $A(\text{'is bounded'}) := 1$ .

The following strings are terms over  $(F, R, A)$ :

+ x y,  
+ ! + x y x,  
! f23.

The following strings are atomic formulae:

'is bounded' + f g,  
< + ! + x y x ! f.

The following strings are formulae:

not 'is bounded' + f g,  
for some f not 'is bounded' + f g,  
or for some f not 'is bounded' + f g < f g.

If not otherwise stated, we will let range " $v, w$ " over variables, " $f$ " over function symbols, " $s, t$ " over terms, " $r$ " over relation symbols, " $a$ " over atomic formulae, and " $p, q$ " over formulae (propositions).

**Exercise 2.8. (Unique Parsing<sup>①</sup> of Polish Expressions)** Show that terms and formulae of the kernel language can be uniquely parsed. *This is done by induction over the terms and formulae.*

### 2.4.2 Extensions of the Kernel Language

We will now introduce some extensions of the kernel language that make formulae easier to read and more compact. Since the extended language is only

used in example formulae we will not bother with defining a complete syntax for the extended language. Rather we will informally, and sometimes only by examples, specify some classes of admissible formulae in the extended language and specify which formulae of the kernel language they abbreviate.

First, we introduce the parentheses "(" and ")" and the comma "," for structuring terms and atomic formulae. For example,

$+ ( x, y )$  stands for  $+ x y$ , and

$< ( x, y )$  stands for  $< x y$ .

Some of the unary function and relation constants may be declared "postfix". For example,

$2 !$  stands for  $! ( 2 )$ .

Some of the binary function and relation symbols may be declared "infix". For example,

$x + y$  stands for  $+ ( x, y )$ .

Some of the binary function and relation symbols may be declared "embracing". For example,

$| x |$  stands for  $|| ( x )$ .

(So far we used spaces between symbols for separating them. From now on we will often omit the spaces as long as it is clear what constitutes a symbol.)

Furthermore, we use English words for the propositional connectives and the existential quantifier. Also we introduce new connectives and quantifiers and different words for existing connectives and quantifiers. Parentheses are also used in more complex formulae for determining grouping of subformulae.

$(p_1 \text{ or } p_2)$  stands for  $\text{or } p_1 p_2$ ,

$(p_1 \text{ and } p_2)$  stands for  $\text{not } (\text{not } p_1 \text{ or } \text{not } p_2)$ ,

$(p_1 \text{ implies } p_2)$  stands for  $(\text{not } p_1 \text{ or } p_2)$ ,

$(p_1 \text{ iff } p_2)$  stands for  $((p_1 \text{ implies } p_2) \text{ and } (p_2 \text{ implies } p_1))$ ,

$\text{for all } v p$  stands for  $\text{not for some } v (\text{not } p)$ ,

$x \not\circ y$  stands for  $\text{not } x \circ y$  (where  $\circ$  is some binary relation constant),

$(p_1 \text{ and } p_2 \text{ and } p_3)$  stands for  
 $(p_1 \text{ and } (p_2 \text{ and } p_3))$ ,

$(p_1 \text{ or } p_2 \text{ or } p_3)$  stands for

$(p_1 \text{ or } (p_2 \text{ or } p_3)),$

$(p_1, p_2, p_3)$  stands for  $(p_1 \text{ and } p_2 \text{ and } p_3).$

if  $p_1$  then  $p_2$  stands for  $p_1$  implies  $p_2,$

if  $p$  then  $q_1$  else  $q_2$  stands for  
 $((\text{if } p \text{ then } q_1) \text{ and } (\text{if not } p \text{ then } q_2)).$

Conjunctions of atomic formulae involving infix relation constants can be contracted. For example,

$x < y < z \leq 1$  stands for  $x < y$  and  $y < z$  and  $z \leq 1.$

Another form of contraction is as follows:

$x, y, z < 1$  stands for  $x < 1$  and  $y < 1$  and  $z < 1.$

Here are some more variants for quantifiers. For example,

for all  $v, w$   $p$  stands for  
 for all  $v$  for all  $w$   $p,$

for all  $v$  with  $p_1$  we have  $p_2$  stands for  
 for all  $v$   $(p_1 \text{ implies } p_2),$

for some  $v$  with  $p_1$  we have  $p_2$  stands for for some  $v$   $(p_1 \text{ and } p_2),$

for some  $v < 1$  we have  $p$  stands for  
 for some  $v$  with  $v < 1$  we have  $p,$

there exists  $v$  such that  $p$  stands for for some  $v$   $p.$

In fact, we will allow a number of different forms of the "key words" for all, there exists, etc. to resemble natural language grammar. For example, sometimes we may want to use there exists a or there exists an instead of there exists etc.

Furthermore, we normally present complex formulae using indentation for increasing readability and also for saving some parentheses. Moving to the next line and going to the right (left) by one tabular position corresponds to one left (right) parenthesis. At the end of such a formula we tacitly assume that all parentheses are closed, i.e. that we go back to the left margin. Also, by using indentation, we sometimes may save words like we have in connection with quantifiers etc.

Given a formula  $p$  of the extended language,  $p^*$  will denote the corresponding formula in the kernel language for which  $p$  is an abbreviation.

**Example 2.9** (A Formula in the Extended Language) Here is the definition of "is continuous" in the extended language of predicate logic:

$f$  'is continuous at'  $x$  iff  
 for all  $\epsilon > 0$   
 there exists a  $\delta > 0$  such that  
 for all  $y$  with  $|y - x| < \delta$  we have  
 $|f(y) - f(x)| < \epsilon$

Note that a couple of function and relation constants are used in this formula:  $-$ ,  $\odot$  are binary function constants,  $||$  is a unary function constant,  $0$  is a 0-ary function constant and 'is continuous',  $>$ , and  $<$  are binary relation constants.  $f$ ,  $x$ ,  $\epsilon$ ,  $\delta$ ,  $y$  are supposed to be in  $V$  in this example.

**Exercise 2.10.** (Polish Notation) For some part  $f$  of the formula in Example 2.9 determine  $f^*$ .

*Solution:* We consider, for example, the formula

there exists a  $\delta > 0$  such that  
 for all  $y$  with  $|y - x| < \delta$  we have  
 $|f(y) - f(x)| < \epsilon$ .

In Polish notation it reads

'for some'  $\delta$  not or not  $> \delta$   $0$  not  
 not 'for some'  $y$  not or not  $< || - y x \delta$   
 $< || - \odot f y \odot f x \epsilon$ .

**Exercise 2.11.** (Grammar for Predicate Logic) Formulate a BNF-grammar for part of the above syntax of the predicate logic extended language. Use the non-terminals "F", "Q" etc. for the set of formulae, quantifier formulae etc. "V" may be used for the set of variables and "A" for the set of atomic formulae. Try to present a grammar that saves parentheses in formulae by exhibiting different "binding strength" for connectives etc. A grammar for the complete extended language would be a major project.

*Solution:*

$$F = Q$$

$$Q = P \mid \text{for all } V \mid \text{for some } V \mid Q$$

$$P = I \mid P \text{ iff } I$$

$$I = D \mid I \text{ implies } D$$

$$D = C \mid D \text{ or } C$$

$$C = B \mid C \text{ and } B$$

$$B = A \mid \text{not } B \mid (Q)$$

□

For practical purposes, it is also very important to have "typed variables" available. A typed variable is a variable for which a "type" (a property of the objects over which the variable ranges) is declared. Typed variables allow to shorten formulae because, by just mentioning a typed variable at a given

position in a formula, the complete type information is supposed to be inserted at this position.

For example, after having introduced the type declaration

typed variables  $\epsilon, \delta$  with range  $\epsilon, \delta \in \mathbb{R}$

the formula

for all  $\epsilon$   
 there exists a  $\delta$  such that  $\text{rel}(\epsilon, \delta)$

is an abbreviation for

for all  $\epsilon \in \mathbb{R}$   
 there exists a  $\delta \in \mathbb{R}$  such that  $\text{rel}(\epsilon, \delta)$ .

A type declaration has a certain "scope". All formulae in the scope of a declaration abbreviate formulae in the untyped language according to the translation process sketched above. In our sloppy version of the extended language we do not introduce a syntax for indicating scopes but assume that the scope extends over all subsequent formulae.

The basic syntax of a type declaration is as follows

typed variable  $v_1$  with range  $p_1$ ,  
 typed variable  $v_2$  with range  $p_2$ ,  
 ...  
 typed variable  $v_n$  with range  $p_n$ .

We will also admit several variants of such declarations, for example

typed variables  $v, w$  with range  $p$

as an abbreviation for

typed variable  $v$  with range  $p$ ,  
 typed variable  $w$  with range  $p$ .

In the scope of the type declaration

typed variable  $v$  with range  $p_1$

the following rules must be applied

for all  $v$   $p_2$  stands for for all  $v$  with  $p_1$  we have  $p_2$ ,

for some  $v$   $p_2$  stands for for some  $v$  with  $p_1$  we have  $p_2$ ,

$p_2$  (with "free variable"  $v$ ) stands for if  $p_1$  then  $p_2$ .

(The notion of a "free variable" will be explained in the next subsection).

In the sequel we will define many notions for terms and formulae. It is understood, that these notions when applied to terms  $t$  and formulae  $p$  of the extended language refer to the abbreviated terms  $t^*$  and formulae  $p^*$  of the kernel language. For example, when we have defined what it means that a variable  $v$  is free in a formula  $p$  of the kernel language then " $v$  is free in the formula  $p$  of the extended language" actually means that " $v$  is free in  $p^*$ ".

### 2.4.3 Free and Bound Variables and Substitution

We will first formally define the concept of "free" and "bound" variables for formulae in the kernel language. This definition and, in fact, most of the subsequent definitions, are "inductive" ("recursive") over the structure of terms and formulae in correspondence to the inductive nature of the definition of the concepts "term" and "formula".

**Definition 2.12. (Free and Bound Variables)** " $v$  is free (bound) in  $t$  ( $p$ )" is inductively defined as follows:

- (1)  $v$  is free in  $w$  iff  $v$  is ~~distinct from~~  $w$ . *(base case of induction)*
  - (2)  $v$  is free in  $f t_1 \dots t_n$  iff  $v$  is free in  $t_1$  <sup>or</sup>  $\dots$  <sup>or</sup>  $t_n$ .
  - (3)  $v$  is free in  $r t_1 \dots t_n$  iff  $v$  is free in  $t_1$  <sup>or</sup>  $\dots$  <sup>or</sup>  $t_n$ .
  - (4)  $v$  is not bound in  $t$ .
  - (5)  $v$  is not bound in  $a$ .
- } *Terms and atomic formulae do not have quantifiers!*
- (6, 6')  $v$  is free (bound) in  $\text{not } p$  iff  $v$  is free (bound) in  $p$ .
  - (7, 7')  $v$  is free (bound) in  $\text{or } p_1 p_2$  iff  $v$  is free (bound) in  $p_1$  or  $v$  is free (bound) in  $p_2$ .
  - (8)  $v$  is not free in  $\text{for some } v p$ .
  - (9)  $v$  is bound in  $\text{for some } v p$ .
  - (10, 10') If  $w$  is distinct from  $v$  then  $v$  is free (bound) in  $\text{for some } w p$  iff  $v$  is free (bound) in  $p$ .

**Example 2.13 (Free and Bound Variables)** The variable  $y$  is bound in  $\text{for some } y > y x$ . By contrast,  $x$  is free in  $\text{for some } y > y x$ . Note that  $x$  is both free and bound in  $\text{or for some } y > y x \text{ for some } x = x x$ . *By rules 7a and 7b.*

In the formula

there exists a  $\delta > 0$  such that  
 for all  $y$  with  $|y - x| < \delta$  we have  
 $|f \odot y - f \odot x| < \epsilon$

$\Delta$  Thus we see that "free" is not the contrary of "bound".



the variables delta and y are bound and the variables x, f and epsilon are free. □

Substitution of terms for free variables is an elementary process for deriving new formulae from given ones. It will be important in the formulation of the rules of inferences for predicate logic.

**Definition 2.14. (Substitution)**  $s_v[t]$  ("the term resulting from  $s$  by substituting  $t$  for  $v$ ") and  $p_v[t]$  ("the formula resulting from  $p$  by substituting  $t$  for  $v$ ") are inductively defined as follows:

$$v_v[t] \stackrel{\text{with}}{=} t.$$

If  $w$  is distinct from  $v$  then  $w_v[t] = w$ .

$$(f s_1 \dots s_n)_v[t] = f (s_1)_v[t] \dots (s_n)_v[t].$$

$$(\tau s_1 \dots s_n)_v[t] = \tau (s_1)_v[t] \dots (s_n)_v[t].$$

$$(\text{not } p)_v[t] = \text{not } p_v[t].$$

$$(\text{or } p_1 p_2)_v[t] = \text{or } (p_1)_v[t] (p_2)_v[t].$$

$$(\text{for some } v p)_v[t] = \text{for some } v p.$$

If  $w$  is distinct from  $v$  then  $(\text{for some } w p)_v[t] = \text{for some } w p_v[t]$ . □

Similarly, for distinct variables  $v_1, \dots, v_n$ , one can define  $s_{v_1, \dots, v_n}[t_1, \dots, t_n]$  and  $p_{v_1, \dots, v_n}[t_1, \dots, t_n]$  ("parallel substitution of  $t_1, \dots, t_n$ ").

† **Exercise 2.15. (Parallel and Sequential Substitution)** Inductively define

$$s_{v_1, \dots, v_n}[t_1, \dots, t_n]$$

and

$$p_{v_1, \dots, v_n}[t_1, \dots, t_n].$$

Give an example where  $s_{v_1, v_2}[t_1, t_2] \neq (s_{v_1}[t_1])_{v_2}[t_2]$ .

**Solution:** The inductive definition of parallel substitution results from the definition of  $s_v[t]$  and  $p_v[t]$  by replacing " $v[t]$ " by " $v_{v_1, \dots, v_n}[t_1, \dots, t_n]$ ". Only the clause for variable terms needs some care:

$$\text{If } v = v_i \text{ then } v_{v_1, \dots, v_n}[t_1, \dots, t_n] = t_i.$$

$$\text{If, for all } i \text{ with } 1 \leq i \leq n, v \neq v_i \text{ then } v_{v_1, \dots, v_n}[t_1, \dots, t_n] = v.$$

An example where parallel substitution is different from "sequential" substitution is

*Can be defined straight-forward in Metamath because it handles pattern matching (expressions may be used as formal parameters). In Lisp one has to use lists where the first element is a "calculator" upon which an if-statement branches to an appropriate part: ?f(x...) = a.*

*But not Metamath:*

*f(x) := Sum(x^2, 1, 2, 3)*

*f(x) will evaluate 5!*

*(Although the wanted "go iterators to avoid local")*

*The total result of  $s_{v_1, v_2}[t_1, t_2]$  when  $v_1$  is different from  $v_2$  is  $s_{v_1}[t_1]_{v_2}[t_2]$ . So it is better to write  $s_{v_1, v_2}[t_1, t_2]$ .*

$$(x < y)_{x,y} [y, 2] = y < 2,$$

$$(x < y)_{x,y} [y]_{y[2]} = 2 < 2.$$

*Mathematically:*

" $x < y$ " i.  $(x \rightarrow y, y \rightarrow 2)$  yields " $y < 2$ "

" $x < y$ " i.  $x \rightarrow y$  i.  $y \rightarrow 2$  yields "false" □

We now must clarify a subtle point in the process of substitution. Our intuition about substitution is that the statement  $p_v[t]$  "says the same thing about the individual denoted by  $t$ " as  $p$  says about the individual denoted by  $v$ . However, if we apply substitution without any precaution, this is not always the case. For example,  $p =$  for some  $y$  ( $x = 2 \cdot y$ ) (under the usual interpretation of the function symbol  $\cdot$ ) says that the individual denoted by  $x$  is even. However,  $p_x[y+1]$ , which is for some  $y$  ( $y+1 = 2 \cdot y$ ), does not say that the individual denoted by  $y+1$  is even but asserts that there is an individual that satisfies the equation  $y+1 = 2 \cdot y$ , i.e. the equation  $y = 1$ . This undesirable effect results from the fact that the variable  $y$  that is free in  $y+1$  becomes bound after substitution into for some  $y$  ( $x = 2 \cdot y$ ). The following definition determines exactly in which situations we would like to allow substitution.

**Definition 2.16. (Substitutable Terms)** " $t$  is substitutable for  $v$  in  $p$ " is inductively defined as follows:

$t$  is substitutable for  $v$  in  $a$ .

$t$  is substitutable for  $v$  in not  $p$  iff  $t$  is substitutable for  $v$  in  $p$ .

$t$  is substitutable for  $v$  in  $or\ p_1\ p_2$  iff  
 $t$  is substitutable for  $v$  in  $p_1$  and in  $p_2$ .

$t$  is substitutable for  $v$  in for some  $v\ p$ . (By our definition of substitution, nothing will happen.)

△ If  $w$  is distinct from  $v$   
 then  $t$  is substitutable for  $v$  in for some  $w\ p$  iff  
 not ( $v$  is free in  $p$  and  $w$  is free in  $t$ ) and  
 $t$  is substitutable for  $v$  in  $p$ .

$$(\forall x (x = 2x))_{y[2]} = \forall x$$

Here:  $t \rightarrow 2x$   
 $v \rightarrow y$   
 $p \rightarrow (\forall y = 2x + y)$   
 $w \rightarrow x$

† **Exercise 2.17. (Substitutable Terms)** Determine whether or not the following terms are substitutable for the following formulae:

Term	Variable	Formula
(1) $y + 1$	$x$	for some $y$ ( $x = 2 \cdot y$ )
(2) $y + 1$	$y$	for some $y$ ( $x = 2 \cdot y$ )
(3) $v \cdot w$	$x$	for some $y$ $x < v \cdot x$ implies for some $w$ ( $w < v$ )
(4) $v \cdot w$	$v$	for some $y$ $x < v \cdot x$ implies for some $w$ ( $w < v$ )

- (5)  $v.w$   $w$  for some  $y$   
 $x < v.x$  implies for some  $w$  ( $w < v$ )

*Solution:*

- (1) No:  $x$  is free in  $x = 2.y$  and  $y$  is free in  $y + 1$ .  
 (2) Yes: Any  $t$  is substitutable for  $y$  in for some  $y p$ .  
 (3) Yes:  $x$  is free in  $x < v.x$  implies for some  $w$  ( $w < v$ ) but  $y$  is not free in  $v.w$  and  $v.w$  is substitutable for  $x$  in for some  $w$  ( $w < v$ ) because  $x$  is not free in  $w < v$ .  
 (4) No:  $v$  is free in  $w < v$  and  $w$  is free in  $v.w$ .  
 (5) Yes:  $w$  is not free in  $x < v.x$  implies for some  $w$  ( $w < v$ ).

In the analysis of the cases (3) - (5) we used the straightforward extension of substitutibility to formulae in the extended language, see next exercise.

**Exercise 2.18. (Substitutibility in the Extended Language)** Show the following rules:

$t$  is substitutable for  $v$  in  $p_1$  and  $p_2$  iff  
 $t$  is substitutable for  $v$  in  $p_1$  and in  $p_2$ .

(Similarly for implies and iff.)

$t$  is substitutable for  $v$  in for all  $v p$ .

If  $w$  is distinct from  $v$   
 then  $t$  is substitutable for  $v$  in for all  $w p$  iff  
 not ( $v$  is free in  $p$  and  $w$  is free in  $t$ ) and  
 $t$  is substitutable for  $v$  in  $p$ .

*Solution:*

and:

$t$  is substitutable for  $v$  in  $p_1$  and  $p_2$   
 iff  
 $t$  is substitutable for  $v$  in not ( $\text{not } p_1$  or not  $p_2$ )  
 iff  
 $t$  is substitutable for  $v$  in ( $\text{not } p_1$  or not  $p_2$ )  
 iff  
 $t$  is substitutable for  $v$  in not  $p_1$  and in not  $p_2$   
 iff  
 $t$  is substitutable for  $v$  in  $p_1$  and in  $p_2$ .

for all, equal variables:

$t$  is substitutable for  $v$  in for all  $v p$   
iff  
 $t$  is substitutable for  $v$  in not for some  $v$  not  $p$   
iff  
 $t$  is substitutable for  $v$  in for some  $v$  not  $p$   
iff  
true.

for all, distinct variables: Let  $w$  be distinct from  $v$ . Then

$t$  is substitutable for  $v$  in for all  $w p$   
iff  
 $t$  is substitutable for  $v$  in not for some  $w$  not  $p$   
iff  
 $t$  is substitutable for  $v$  in for some  $w$  not  $p$   
iff  
not ( $v$  is free in not  $p$  and  $w$  is free in  $t$ ) and  
 $t$  is substitutable for  $v$  in not  $p$   
iff  
not ( $v$  is free in  $p$  and  $w$  is free in  $t$ ) and  
 $t$  is substitutable for  $v$  in  $p$ .

In the sequel, if  $p_v[t]$  appears somewhere in a statement we tacitly assume that  $p, v, t$  represent expressions such that  $t$  is substitutable for  $v$  in  $p$ .

The concept of substitutibility is important for all formal systems involving variables. For example, it also plays an important role in Church's " $\lambda$ -calculus". Since substitutibility is fairly complicated, some formal systems try to get along without any variables at all, for example Curry's "logic of combinators". That this is possible is a surprising result.

## 2.5 The Semantics of First Order Predicate Logic

Given an *interpretation* of function and predicate symbols in a *domain*, a first order variable-free *term* denotes an object in the domain (the object is the "value" of the term) and a *closed formula* (i. e. a formula without free variables) denotes a fact about the domain that can be true or false ("true" or "false" is the "truth value" of the formula). In the subsequent subsections we will define the concepts involved in this view of semantics, i.e. the concepts of

1. domain,
2. interpretation, and
3. ~~and~~ value (denotation).

This will finally enable us to define what it means that a formula is a *logical consequence* of other formulae.

### 2.5.1 Domains

We assume that the concepts of "tuples" and "Cartesian product of sets" are already available. Thus, we do not define these concepts here but only summarize our notation. For arbitrary  $n \in \mathbb{N}_0$ , we will write  $(m_1, \dots, m_n)$  for the  $n$ -tuple formed from the objects  $m_1, \dots, m_n$ . In particular,  $()$  is the 0-tuple. The Cartesian product of  $M_1, \dots, M_n$ , i.e. the set  $\{(m_1, \dots, m_n) \mid m_1 \in M_1, \dots, m_n \in M_n\}$ , is denoted by  $M_1 \times \dots \times M_n$ .  $M^n$  is an abbreviation for  $\underbrace{M \times \dots \times M}_{n \text{ times}}$ .

$M^0$  is the set  $\{()\}$ , a singleton.

#### Definition 2.19. (Functions and Relations)

$\phi: A \rightarrow B$  (" $\phi$  is a mapping from  $A$  to  $B$ ") iff

*nota*  $\leftarrow \phi \subseteq A \times B$  and

for all  $x \in A$ , there exists a unique  $y \in B$  such that  $(x, y) \in \phi$ .

$\hookrightarrow$  "left-total" (surjektive)  $\hookrightarrow$  "right-unique" (injektiv)

If  $\phi: A \rightarrow B$  and  $x \in A$  then  $\phi(x) :=$  the  $y$  such that  $(x, y) \in \phi$ .

$\phi$  is a mapping iff  $\phi: A \rightarrow B$  for some  $A, B$ .

If  $\phi$  is a mapping then

the *domain* of  $\phi := \{x \mid \text{for some } y, (x, y) \in \phi\}$ ,

the *range* of  $\phi := \{y \mid \text{for some } x, (x, y) \in \phi\}$ .

A mapping  $\phi$  is *injective* iff

for all  $x_1, x_2, y$ , if  $(x_1, y), (x_2, y) \in \phi$  then  $x_1 = x_2$ .

$\hookrightarrow$  "left-unique"

( $\phi$  surjective (right-total)  
 $\hookrightarrow$  range  $(\phi: A \rightarrow B) = B$ )

$\phi$  is an  $n$ -ary (total) function on  $C$  iff  $\phi: C^n \rightarrow C$ .

$\hookrightarrow$  that means: domain  $(\phi: A \rightarrow B) = A$ , i.e. left-total

$\phi$  is a multi-ary function on  $C$  iff, for some  $n$ ,  $\phi: C^n \rightarrow C$ .

\* For multi-ary functions, the typical mapping that describes a function will not be a mapping.

If  $\phi$  is a multi-ary function on  $C$   
 then  $\text{arity}(\phi) :=$  the  $n$  such that  $\phi: C^n \rightarrow C$ .

$\rho$  is an  $n$ -ary relation on  $C$  iff  $\rho \subseteq C^n$ .

$\rho$  is a multi-ary relation on  $C$  iff, for some  $n$ ,  $\rho \subseteq C^n$ .

If  $\rho$  is a multi-ary relation on  $C$   
 then  $\text{arity}(\rho) :=$  the  $n$  such that  $\rho \subseteq C^n$ . □

Notation: We write  $\phi(c_1, \dots, c_n)$  instead of  $\phi((c_1, \dots, c_n))$ .

**Definition 2.20. (Domains)**  
 (= structure, universe)

$\Delta = (\Gamma, \Phi, P)$  is a (homogeneous) domain (or structure) iff  
 $\Gamma$  is a non-empty set, ("carrier", "Trägermenge")  
 $\Phi$  is a set of multi-ary functions on  $\Gamma$  and  
 $P$  is a set of multi-ary relations on  $\Gamma$ .

Do not confuse with "domain of discourse" (with def. 2.21)!

Very general concept!  
 (Virtually anything 'interesting' is a domain.)



Note that a 0-ary function on  $M$  has the form  $\{(( ), m)\}$  where  $m$  is an element in  $M$ . Thus, the 0-ary functions on  $M$  are in 1-1 correspondence with the elements in  $M$ .

**Example 2.21. (A Finite Domain)** Let  $\Gamma := \{1, 2\}$ ,  $\phi_1 := \{(1, 2), (2, 2)\}$ ,  $\phi_2 := \{(1, 1, 1), (1, 2, 1), (2, 1, 1), (2, 2, 2)\}$ ,  $\rho := \{(1, 2), (1, 1), (2, 2)\}$ .  $\phi_1$  is a unary function on  $M$ ,  $\phi_2$  is a binary function on  $M^2$  and  $\rho$  is a binary relation on  $M$ . Let  $\Phi := \{\phi_1, \phi_2\}$  and  $P := \{\rho\}$ .  $\Delta := (\Gamma, \Phi, P)$  is a domain.

**Example 2.22. (An Infinite Domain)** Let  $\Delta := (\mathbb{R}, \{\sin, \cos, \cdot\}, \{\mathbb{N}, <, =\})$ .  $\Delta$  is a domain.  $\sin$  and  $\cos$  are unary functions on  $\mathbb{R}$ ,  $\cdot$  is a binary function on  $\mathbb{R}$ .  $\mathbb{N}$  is a unary relation and  $<$  and  $=$  are binary relations on  $\mathbb{R}$ .

**Example 2.23. (A Heterogeneous Domain)** Domains of non-logical symbols are not homogeneous domains in the sense of the above definition. They are "heterogeneous" domains (having several carrier sets instead of just one). We do not consider the notion of heterogeneous domains any further here.

2.5.2 Interpretations

**Definition 2.24. (Interpretations of First Order Constants)** Let  $S = (F, R, A)$  be a domain of constants and  $\Delta = (\Gamma, \Phi, P)$  a domain.  $I$  is an interpretation of  $S$  in  $\Delta$  iff

$I: (F \cup R) \rightarrow (\Phi \cup P)$ ,  
 for all  $f, I(f) \in \Phi$  and  $A(f) = \text{arity}(I(f))$ ,  
 for all  $r, I(r) \in P$  and  $\text{arity}(r) = A(I(r))$ .

$A(r) = \text{arity}(I(r))$

*f and r are in universe*

2.5.3 Name Generators

Given a domain of constants and a structure, for technical reasons we will need an extra "name" for each individual in the domain. For this we have to extend the domain of constants in a suitable way. We consider name generators that produce sufficiently many names for individuals.

**Definition 2.25. (Name Generators)** Let  $(F, R, A)$  be a domain of constants and  $(\Gamma, \Phi, P)$  a domain. Then  $'$  is a *name generator* for  $\Gamma$  in  $(F, R, A)$  iff

- ' is an injective function,
- the domain of  $'$  is  $\Gamma$ , and
- the range of  $'$  is a set (of symbols) disjoint from  $F \cup R$ . □

Notation: We will write  $c'$  instead of  $'(c)$ . Also, if  $\Gamma$  and  $(F, R, A)$  is clear from the context,  $'$  will always be a name generator for  $\Gamma$  in  $(F, R, A)$ . Furthermore, in such context,  $F'$  will denote  $F \cup \{\gamma' \mid \gamma \in \Gamma\}$ .

**Example 2.26. (A Name Generator)** Let  $\Gamma = \mathbb{N}$  and  $F := \{0, +\}$ ,  $R$  and  $A$  arbitrary. Then  $'$  with  $0' := \text{zero}$ ,  $n' :=$  the decimal number representation of  $n$  in type writer notation (for  $n > 0$ ) would be a possible name generator for  $\Gamma$  in  $(F, R, A)$ . (For example,  $2' = 2$ ). Note that we must not define  $0' := 0$  because then the range of  $'$  would not be disjoint from  $F \cup R$ .  
*It is no harm that we now add the words: "0", "and".*

*It is not necessary (and as usual and not desirable) to  $F = \{0, +, 2, 3, \dots\}$  although  $\Gamma$  now is not by a  $2, 3, \dots$  etc. note:  $\Gamma$  is the universe.*

2.5.4 Values of Terms and Formulae

We define now, first, the values of variable-free terms and the truth values of closed formulae under a given interpretation and, then, the notion of validity of arbitrary formulae under an interpretation.

**Definition 2.27. (Variable-Free Terms and Closed Formulae)**

A term  $t$  over  $S$  is *variable-free* iff, for no  $v$ ,  $v$  is free in  $t$ .

A formula  $p$  over  $S$  is *closed* iff, for no  $v$ ,  $v$  is free in  $p$ . □

Let  $V$  be a set of first order variables,  $S = (F, R, A)$  a domain of constants, and  $\Delta = (\Gamma, \Phi, P)$  a domain. Let  $I$  be an interpretation of  $S$  in  $\Delta$ .

**Definition 2.28. (The Values of Terms)** For all variable-free terms  $t$  over  $S$  we define  $\langle t \rangle_I$  ("the value of  $t$ " or "the value denoted by  $t$  under the interpretation  $I$ ") inductively as follows:

If  $\gamma \in \Gamma$  then  $\langle \gamma' \rangle_I = \gamma$ .

If  $A(f) = n$  then  $\langle f t_1 \dots t_n \rangle_I = I(f)(\langle t_1 \rangle_I, \dots, \langle t_n \rangle_I)$ . □

Once and for all, we choose now two distinct objects  $T$  and  $F$  ("true" and "false") called "truth values".

**Definition 2.29. (Truth Functions)** The following functions are called *truth functions*.

$v$	$w$	$B_{\neg}(v)$	$B_{\wedge}(v, w)$	$B_{\vee}(v, w)$	$B_{\Rightarrow}(v, w)$	$B_{\Leftrightarrow}(v, w)$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

□

The truth functions reflect the intuitive meaning of the propositional connectives. In the following definition they are used for describing precisely how the truth values of compound propositional formulae are determined from the truth values of the constituents.

**Definition 2.30. (The Truth Values of Closed Formulae Under Interpretations)** For all closed formulae  $p$  over  $S$  we define  $\langle p \rangle_{I, \Delta}$  ("the truth value of  $p$ " or "the truth value denoted by  $p$  under the interpretation  $I$  in  $\Delta$ ") or, short,  $\langle p \rangle_I$  (if  $\Delta$  is clear from the context) inductively as follows:

$$\langle t_1 = t_2 \rangle_I = \text{T iff } \langle t_1 \rangle_I = \langle t_2 \rangle_I.$$

*"F otherwise" (always to be added)*

If  $A(r) = n$  (and  $r \neq =$ ) then  
 $\langle r t_1 \dots t_n \rangle_I = \text{T iff } (\langle t_1 \rangle_I, \dots, \langle t_n \rangle_I) \in I(r).$

$$\langle \text{not } p \rangle_I = B_{\neg}(\langle p \rangle_I).$$

$$\langle \text{or } p_1 p_2 \rangle_I = B_{\vee}(\langle p_1 \rangle_I, \langle p_2 \rangle_I).$$

$$\langle \text{for some } v p \rangle_I = \text{T iff, for some } \gamma \in \Gamma, \langle p_v[\gamma] \rangle_I = \text{T}.$$

**Example 2.31. (Values of Terms and Formulae)** Let  $V := \{x, y\}$ ,  $S := (F, R, A)$ , where  $F := \{+, \cdot\}$ ,  $R := \{<\}$ ,  $A(\cdot) := A(<) := 2$ ,  $A(+)$  := 1. Let  $t := + \cdot 2' 1'$  and  $p := \text{'for some' } y < + \cdot 2' y 2'$ .

Let  $\Delta$  be defined as in Example 2.21.

Let  $I(+)$  :=  $\phi_1$ ,  $I(\cdot)$  :=  $\phi_2$ ,  $I(<) := \rho$ .  $I$  is an interpretation of  $S$  in  $\Delta$ . We have  $\langle t \rangle_I = \phi_1(\phi_2(2, 1)) = 2$ . Furthermore,  $\langle < + \cdot 2' 1' 2' \rangle_I = \text{T}$  because  $(2, 2) \in \rho$ . This shows that  $\langle p \rangle_I = \text{T}$  because  $\langle < + \cdot 2' y 2' y [1'] \rangle_I = \text{T}$ .

### 2.5.5 Validity, Models, Logical Consequence, Equivalence, Satisfiability

**Definition 2.32. (Validity Under Interpretations)** Let  $p$  be a formula over  $S$ .

If  $p$  is closed then  
 $p$  is valid under the interpretation  $I$  in the domain  $\Delta$  iff  $\langle p \rangle_{I, \Delta} = \text{T}$ .

If  $p$  contains exactly the <sup>distinct</sup> free variables  $v_1, \dots, v_n$  then  
 $p$  is valid under the interpretation  $I$  in  $\Delta$  iff  
 for all  $\gamma_1, \dots, \gamma_n \in \Gamma$ ,  $\langle p_{v_1, \dots, v_n}[\gamma_1, \dots, \gamma_n] \rangle_{I, \Delta} = \text{T}$ .

↑  
*for all interpretations*



**Example 2.33** (Valid Formulae under Interpretations) Let  $S, \Delta$  and  $I$  be as defined in the last example and let  $p := \text{'for some } y < x \text{ } y. p \text{'}$  is valid under  $I$  because  $\langle p_x[1'] \rangle_I = \mathbf{T}$  and  $\langle p_x[2'] \rangle_I = \mathbf{T}$ . (Note that  $(1, 2) \in \rho$  and  $(2, 2) \in \rho$ .)

**Definition 2.34.** (Logical Validity)

$p$  is valid in predicate logic (or just valid or a predicate logical tautology) iff, for all domains  $\Delta$  and for all interpretations  $I$  in  $\Delta$ ,  $p$  is valid under  $I$  in  $\Delta$ . □

**Example 2.35** (Valid Formulae)  $(x = x)$  or not  $(x = x)$  is a logically valid formula because, whatever the value  $v := \langle (x = x)_x[\gamma'] \rangle_I$  is, the value  $\langle (x = x) \text{ or not } (x = x) \rangle_x[\gamma']_I = B_{\vee}(v, B_{\neg}(v)) = \mathbf{T}$ .

**Exercise 2.36.** (Truth Values for Extended Formulae) Show that

$$\langle \text{and } p_1 \text{ } p_2 \rangle_I = B_{\wedge}(\langle p_1 \rangle_I, \langle p_2 \rangle_I).$$

$$\langle \text{implies } p_1 \text{ } p_2 \rangle_I = B_{\Rightarrow}(\langle p_1 \rangle_I, \langle p_2 \rangle_I).$$

$$\langle \text{iff } p_1 \text{ } p_2 \rangle_I = B_{\Leftrightarrow}(\langle p_1 \rangle_I, \langle p_2 \rangle_I).$$

$$\langle \text{for all } v \text{ } p \rangle_I = \mathbf{T} \text{ iff, for all } \gamma \in \Gamma, \langle p_v[\gamma'] \rangle_I = \mathbf{T}.$$

*Solution:*

and:

$$\begin{aligned} \langle \text{and } p_1 \text{ } p_2 \rangle_I & \stackrel{\text{Def. of 'and'}}{=} \\ & \stackrel{\text{Def. of } \langle \cdot \wedge \cdot \rangle, \langle \cdot \Rightarrow \cdot \rangle}{=} \\ & = \langle \text{not (not } p_1 \text{ or not } p_2) \rangle_I \\ & \stackrel{\text{see above}}{=} \\ & = B_{\neg}(B_{\vee}(B_{\neg}(\langle p_1 \rangle_I), B_{\neg}(\langle p_2 \rangle_I))) \\ & = B_{\wedge}(\langle p_1 \rangle_I, \langle p_2 \rangle_I). \end{aligned}$$

The last equality in this derivation holds because for arbitrary truth values  $v, w \in \{\mathbf{T}, \mathbf{F}\}$  we have

$v$	$w$	$v_1 := B_{\neg}(v)$	$w_1 := B_{\neg}(w)$	$u := B_{\vee}(v_1, w_1)$	$B_{\neg}(u)$	$B_{\wedge}(v, w)$
<b>T</b>	<b>T</b>	<b>F</b>	<b>F</b>	<b>F</b>	<b>T</b>	<b>T</b>
<b>T</b>	<b>F</b>	<b>F</b>	<b>T</b>	<b>T</b>	<b>F</b>	<b>F</b>
<b>F</b>	<b>T</b>	<b>T</b>	<b>F</b>	<b>T</b>	<b>F</b>	<b>F</b>
<b>F</b>	<b>F</b>	<b>T</b>	<b>T</b>	<b>T</b>	<b>F</b>	<b>F</b>

The last two columns are identical.

implies and iff: The proof is similar.

for all:

- $\langle \text{for all } v \ p \rangle_I = \mathbf{T}$   
 iff  $\leftarrow$  def. of "for all".  
 $\langle \text{not for some } v \ \text{not } p \rangle_I = \mathbf{T}$   
 iff  $\leftarrow$  def. of  $\langle \text{not} \rangle$ .  
 $\langle \text{for some } v \ \text{not } p \rangle_I = \mathbf{F}$   
 iff  $\leftarrow$  def. of  $\langle \text{for some} \rangle$ .  
 $\langle (\text{not } p)_v[\gamma'] \rangle_I = \mathbf{T}$  for no  $\gamma \in \Gamma$   
 iff  $\leftarrow$  Inference rule (De Morgan's rule on meta level + Def. of substitution ("not"), 6.7).  
 $\langle \text{not } (p_v[\gamma']) \rangle_I = \mathbf{F}$  for all  $\gamma \in \Gamma$   
 iff  $\leftarrow$  def. of  $\langle \text{not} \rangle$ .  
 $\langle (p_v[\gamma']) \rangle_I = \mathbf{T}$  for all  $\gamma \in \Gamma$ .

**Definition 2.37. (Closure of Formulae)**

If all free variables of  $p$  are among the distinct variables  $v_1, \dots, v_n$  then  $\bar{p}$  is a closure of  $p$  iff  $\bar{p} = \text{for all } v_1, \dots, v_n \ p$ .

**Exercise 2.38. (Closure and Validity)** Let  $\bar{p}$  be a closure of  $p$ . Show that

$p$  is valid under  $I$  in  $\Delta$  iff  $\langle \bar{p} \rangle_{I, \Delta} = \mathbf{T}$  (i. e.  $\bar{p}$  is valid under  $I$  in  $\Delta$ ).

**Solution:** We show the proposition for the case where  $p$  contains not more than two free variables  $v_1$  and  $v_2$  and  $\bar{p} = \text{for all } v_1, v_2 \ p$ .

*(simplest non-trivial case (complication - see comment below))*  
 $p$  is valid under  $I$  in  $\Delta$

- iff  $\leftarrow$  def. of validity.  
 for all  $\gamma_1, \gamma_2 \in \Gamma$ ,  $\langle p_{v_1, v_2}[\gamma'_1, \gamma'_2] \rangle_{I, \Delta} = \mathbf{T}$   
 iff  $\leftarrow$  see comment below.  
 for all  $\gamma_1, \gamma_2 \in \Gamma$ ,  $\langle (p_{v_1}[\gamma'_1])_{v_2}[\gamma'_2] \rangle_{I, \Delta} = \mathbf{T}$   
 iff  $\leftarrow$  rule for  $\langle \text{for all} \rangle$  (see exercise 2.36).  
 for all  $\gamma_1 \in \Gamma$ ,  $\langle \text{for all } v_2 \ (p_{v_1}[\gamma'_1]) \rangle_{I, \Delta} = \mathbf{T}$   
 iff  $\leftarrow$  definition of substitution ("for all" ...), [1].  
 for all  $\gamma_1 \in \Gamma$ ,  $\langle (\text{for all } v_2 \ p)_{v_1}[\gamma'_1] \rangle_{I, \Delta} = \mathbf{T}$   
 iff  $\leftarrow$  rule for  $\langle \text{for all} \rangle$  (see exercise 2.10).  
 $\langle \text{for all } v_1 \ \text{for all } v_2 \ p \rangle_{I, \Delta} = \mathbf{T}$   
 iff  $\leftarrow$  def. of  $\bar{p}$ .  
 $\langle \bar{p} \rangle_{I, \Delta} = \mathbf{T}$ .

In this derivation it is important to note that, since  $\gamma'_1$  is a constant, parallel substitution has the same effect as sequential substitution. Also note that the proposition is also true in the case where  $v_1$  or  $v_2$  does not actually occur in  $p$ .

□

The last proposition shows that, in the formulation of predicate logic we consider here, free variables "have the same meaning" as universally quantified variables.

**Definition 2.39. (Models)** Let  $I$  be an interpretation in a domain  $\Delta$  and  $Q$  a set of formulae.

*Also called "model"*

$\Delta$  is a *model* of  $Q$  under the interpretation  $I$  iff  
all  $q \in Q$  are valid under  $I$  in  $\Delta$ .  $\square$

Convention: We say " $\Delta$  is a model for  $p$  under  $I$ " instead of " $\Delta$  is a model for  $\{p\}$  under  $I$ ".

**Example 2.40. (Models)** Let  $Q$  consist of the three formulae

$$\begin{aligned}x \cdot (y \cdot z) &= (x \cdot y) \cdot z, \\1 \cdot x &= x, \\x' \cdot x &= 1.\end{aligned}$$

where  $\cdot$ ,  $'$ , and  $1$  are binary, unary and 0-ary function symbols respectively. These formulae are called the group axioms.

Let  $\Delta = (\mathbb{R}_{n,n}, \{ \cdot, E_{n,n}, {}^{-1} \}, \{ \})$ , where  $\mathbb{R}_{n,n}$  is the set of all regular  $n$  by  $n$  matrices over  $\mathbb{R}$ ,  $\cdot$  is matrix multiplication,  $E_{n,n}$  is the  $n$  by  $n$  unity matrix, and  ${}^{-1}$  is matrix inversion. Let furthermore  $I(\cdot) = \cdot$ ,  $I(E_{n,n}) = 1$ , and  $I({}^{-1}) = {}^{-1}$ . Now,  $\Delta$  is a model of the set of group axioms under the interpretation  $I$ .  $\square$

We now proceed to the rigorous definition of the notion of "logical consequence" that goes back to (Tarski 1935). Intuitively, the main idea in the notion of "logical" consequence is that a formula  $p$  is considered to follow "logically" from a set  $Q$  of formulae if  $p$  is valid whenever  $Q$  is valid, *independent of the meaning* one attaches to the symbols occurring in  $p$  and  $Q$ . Differently stated,  $p$  should be valid for all possible meanings of the symbols for which  $Q$  becomes valid. (Carefully think about this kind of paraphrasing "independent" by "for all"! By analogy, consider a real function that "does not depend" on its argument, i.e. a constant function, i.e. a function that, "for all input values", yields the same output.)

**Definition 2.41. (Predicate Logical Consequences)** Let  $Q$  be a set of formulae.

$Q \models p$  <sup>or semantic</sup>  
("p is a predicate logical consequence of Q" or  
"p is valid in the theory Q") iff  
for all domains  $\Delta$  and for all interpretations  $I$  in  $\Delta$   
if all  $q \in Q$  are valid under  $I$  in  $\Delta$   
then  $p$  is valid under  $I$  in  $\Delta$   
(i.e. if  $\Delta$  is a model for  $Q$  under  $I$   
then  $\Delta$  is a model for  $p$  under  $I$ ).  $\square$

Convention: We write " $q \models p$ " instead of " $\{q\} \models p$ ".

**Example 2.42. (A Predicate Logical Consequence)** Let  $Q$  be the set of group axioms. The models of  $Q$  ~~under the~~ are called "groups". Then the formula

$$p_1 := x \cdot 1 = x$$

is a predicate logical consequence of  $Q$  because  $p_1$  is valid in all groups. By contrast, the formula

$$p_2 := x \cdot y = y \cdot x$$

is not a predicate logical consequence of  $Q$  because there are groups that are non-commutative.

**Exercise 2.43 (Proof of a Predicate Logical Consequences)** Of course, it is not possible to verify that  $x \cdot 1 = x$  is a predicate logical consequence of the group axioms by "observing" its validity in all groups. Rather one must "prove" it. Try to give a proof!

*Solution:*  $x \cdot 1 = x \cdot (x' \cdot x) = (x \cdot x') \cdot x = 1 \cdot x$ .

For this derivation we used that  $x \cdot x' = 1$  is also a predicate logical consequence of the group axioms. Here is the proof of this fact:  $x \cdot x' = 1 \cdot (x \cdot x') = (x'' \cdot x') \cdot (x \cdot x') = x'' \cdot ((x' \cdot x) \cdot x') = x'' \cdot (1 \cdot x') = x'' \cdot x' = 1$ .

**Exercise 2.44 (Predicate Logical Consequence and Validity)** Let  $\bar{q}_1, \dots, \bar{q}_n, \bar{p}$  be closures of  $q_1, \dots, q_n$ . Show that

$p$  is a predicate logical consequence of  $q_1, \dots, q_n$  iff  
 $((\bar{q}_1 \text{ and } \dots \text{ and } \bar{q}_n) \text{ implies } \bar{p})$  is valid. 23

*Solution:*

$p$  is a predicate logical consequence of  $q_1, \dots, q_n$

iff

for all  $\Delta$  and  $I$ ,

if  $q_1, \dots, q_n$  are valid under  $I$  then  $p$  is valid under  $I$

iff

for all  $\Delta$  and  $I$ ,

if  $\langle \bar{q}_1 \rangle_I = \dots = \langle \bar{q}_n \rangle_I = \mathbf{T}$  then  $\langle \bar{p} \rangle_I = \mathbf{T}$ .

iff

for all  $\Delta$  and  $I$ ,

$B \Rightarrow (B \wedge (\langle \bar{q}_1 \rangle_I, \dots, \langle \bar{q}_n \rangle_I), \langle \bar{p} \rangle_I) = \mathbf{T}$

iff

for all  $\Delta$  and  $I$ ,

$((\bar{q}_1 \text{ and } \dots \text{ and } \bar{q}_n) \text{ implies } \bar{p})_I = \mathbf{T}$

iff

$((\bar{q}_1 \text{ and } \dots \text{ and } \bar{q}_n) \text{ implies } \bar{p})$  is valid. n

**Exercise 2.45 (Semantical Modus Ponens)** Show the following statement:

If  $((q_1 \text{ and } \dots \text{ and } q_n) \text{ implies } p)$  is valid and

$q_1, \dots, q_n$  are valid under  $I$  in  $\Delta$

then  $p$  is valid under  $I$  in  $\Delta$ .

*Solution:* For simplicity of notation let us assume that in  $((q_1 \text{ and } \dots \text{ and } q_n) \text{ implies } p)$  we have exactly one free variable  $v$ . Let  $\gamma$  be arbitrary in  $\Gamma$ . We have to show  $\langle p_v[\gamma'] \rangle_I = \mathbf{T}$ . Since  $q_1, \dots, q_n$  are valid under  $I$  in  $\Delta$ , we have  $\langle (q_1)_v[\gamma'] \rangle_I = \dots = \langle (q_n)_v[\gamma'] \rangle_I = \mathbf{T}$ . Since  $((q_1 \text{ and } \dots \text{ and } q_n) \text{ implies } p)$  is a predicate logical tautology, we also know that  $\langle ((q_1 \text{ and } \dots \text{ and } q_n) \text{ implies } p)_v[\gamma'] \rangle_I = \langle ((q_1)_v[\gamma'] \text{ and } \dots \text{ and } (q_n)_v[\gamma']) \text{ implies } p_v[\gamma'] \rangle_I = \mathbf{T}$ . Hence,  $\langle p_v[\gamma'] \rangle_I = \mathbf{T}$ .

**Definition 2.46. (Equivalent Formulae)**

$p$  is predicate logically equivalent to  $q$  iff  $p \models q$  and  $q \models p$ .

**Exercise 2.47. (Characterization of Equivalence)** Show that

$p$  is predicate logically equivalent to  $q$  iff  
 for all domains  $\Delta$  and for all interpretations  $I$  in  $\Delta$   
 $p$  is valid under  $I$  in  $\Delta$  iff  $q$  is valid under  $I$  in  $\Delta$ .

*Solution:* :

$p$  is logically equivalent to  $q$   
 iff  
 $p \models q$  and  $q \models p$   
 iff  
 for all  $\Delta$  and  $I$ , if  $p$  is valid under  $I$  in  $\Delta$  then  $q$  is valid under  $I$  in  $\Delta$  and  
 for all  $\Delta$  and  $I$ , if  $q$  is valid under  $I$  in  $\Delta$  then  $p$  is valid under  $I$  in  $\Delta$   
 iff  
 for all  $\Delta$  and  $I$ ,  $p$  is valid under  $I$  in  $\Delta$  iff  $q$  is valid under  $I$  in  $\Delta$ .

**Example 2.48. (Equivalent Formulae)** For arbitrary  $w$  and  $p$ , not for all  $w$   $p$  is equivalent to for some  $w$  not  $p$ . For simplicity of notation, we show this only in the case where not for all  $w$   $p$  contains exactly one free variable  $v$ . Namely,

not for all  $w$   $p$  is equivalent to for some  $w$  not  $p$   
 iff (by the previous proposition)  
 for all  $\Delta$  and  $I$ ,  
 not for all  $w$   $p$  is valid under  $I$  in  $\Delta$  iff  
 for all  $w$  not  $p$  is valid under  $I$  in  $\Delta$   
 iff  
 for all  $\Delta$  and  $I$ ,  
 for all  $\gamma$ ,  $\langle\langle$ not for all  $w$   $p$  $\rangle\rangle_v[\gamma']_I = \mathbf{T}$  iff  
 for all  $\gamma$ ,  $\langle\langle$ for some  $w$  not  $p$  $\rangle\rangle_v[\gamma']_I = \mathbf{T}$ .

The last statement is true because we even can prove the stronger statement that, for arbitrary  $\gamma$ ,

$\langle\langle$ not for all  $w$   $p$  $\rangle\rangle_v[\gamma']_I = \langle\langle$ for some  $w$  not  $p$  $\rangle\rangle_v[\gamma']_I$ ,

namely,

$\langle\langle$ not for all  $w$   $p$  $\rangle\rangle_v[\gamma']_I = \mathbf{T}$   
 iff  
 $\langle$ not for all  $w$   $p$  $\rangle_v[\gamma']_I = \mathbf{T}$   
 iff  
 $\langle$ for all  $w$   $p$  $\rangle_v[\gamma']_I = \mathbf{F}$   
 iff  
 $\langle\langle$  $p$  $\rangle\rangle_w[\gamma'_1]_I = \mathbf{F}$  for some  $\gamma_1$

iff  
 $\langle \text{not } (p_v[\gamma'])_w[\gamma'_1] \rangle_I = \text{T}$  for some  $\gamma_1$   
 iff  
 $\langle (\text{not } p_v[\gamma'])_w[\gamma'_1] \rangle_I = \text{T}$  for some  $\gamma_1$   
 iff  
 $\langle \text{for some } w \text{ not } p_v[\gamma'] \rangle_I = \text{T}$   
 iff  
 $\langle (\text{for some } w \text{ not } p)_v[\gamma'] \rangle_I = \text{T}$ .

**Definition 2.49. (Satisfiability)**

$p$  is *satisfiable* iff,  
 for some domain  $\Delta$  and some interpretations  $I$  in  $\Delta$ ,  
 $p$  is valid under  $I$  in  $\Delta$ .

$Q$  is *satisfiable* iff,  
 for some domain  $\Delta$  and some interpretations  $I$  in  $\Delta$ ,  
 all  $q \in Q$  are valid under  $I$  in  $\Delta$ .

**Exercise 2.50. (Characterization of Satisfiability)** Let  $\bar{p}$  be a closure of  $p$ . Show that

$p$  is satisfiable  
 iff  
 not  $\bar{p}$  is not valid.

Hence, also

$p$  is valid  
 iff  
 not  $\bar{p}$  is not satisfiable.

*Solution:*

$p$  is satisfiable  
 iff — *Def. of satisfiability*  
 for some  $\Delta$  and  $I$ ,  $p$  is valid in  $\Delta$  under  $I$ .  
 iff — *Ex. 2.49*  
 for some  $\Delta$  and  $I$ ,  $\langle \bar{p} \rangle_I = \text{T}$   
 iff — *Def. of  $\langle \cdot \rangle$ , item "not"*  
 for some  $\Delta$  and  $I$ ,  $\langle \text{not } \bar{p} \rangle_I = \text{F}$   
 iff — *De Morgan's rule on meta level*  
 not for all  $\Delta$  and  $I$ ,  $\langle \text{not } \bar{p} \rangle_I = \text{T}$   
 iff — *Def. of satisfiability for closed formula*  
 not  $\bar{p}$  is not valid.

*Handwritten notes:*  
 $\bar{p}$  is ~~valid~~ not valid  
 $\bar{p}$  is valid not valid  
 $\neg \neg \bar{p}$  is not valid  
 $\bar{p}$  satisf.

**Exercise 2.51. (Predicate Logical Consequence and Unsatisfiability)** Let  $\bar{p}$  be a closure of  $p$ . Show that

$Q \models p$  iff  $Q \cup \{\text{not } \bar{p}\}$  is not satisfiable.

*Hint:*  $p \supset q \equiv \text{not } p \vee q \equiv \neg(p \wedge \neg q)$

*Handwritten note:*  $\neg \bar{p}$  is not satisfiable

*Solution:*

$$Q \models p$$

iff — Def. of  $\models$

for all  $\Delta$  and  $I$ ,

if, for all  $q \in Q$ ,  $q$  is valid under  $I$  in  $\Delta$

then  $p$  is valid under  $I$  in  $\Delta$

iff — 6.2.33

for all  $\Delta$  and  $I$ ,

if, for all  $q \in Q$ ,  $q$  is valid under  $I$  in  $\Delta$

then  $\bar{p}$  is valid under  $I$  in  $\Delta$

iff — De Morgan's rule on meta level

for no  $\Delta$  and  $I$ ,

for all  $q \in Q$ ,  $q$  is valid under  $I$  in  $\Delta$  and

$\bar{p}$  is not valid under  $I$  in  $\Delta$

iff —

for no  $\Delta$  and  $I$ ,

for all  $q \in Q$ ,  $q$  is valid under  $I$  in  $\Delta$  and

not  $\bar{p}$  is valid under  $I$  in  $\Delta$

iff — Def. of satisfiability

$Q \cup \{\text{not } \bar{p}\}$  is not satisfiable.

### 2.5.6 Proposition Logical Semantics

For most formulae  $p$  of predicate logic it is not immediately clear how to determine whether or not  $p$  is valid. However, for some formulae, validity can be determined by merely looking at the structure of the formulae in terms of the propositional connectives  $\neg$  and  $\vee$  (and the other propositional connectives  $\wedge$ ,  $\rightarrow$ , and  $\leftrightarrow$  in the extended language). For example, it is immediately clear that  $(\exists x (x < 0)) \vee \neg (\exists x (x < 0))$  is valid because this formula has the "propositional structure"  $p \vee \neg p$  and, whatever the truth value of  $p$  is,  $p \vee \neg p$  receives the truth value T.

The logic of formulae in terms of their structure with respect to the propositional connectives is called "propositional logic". It is an important part of predicate logic. In some way, propositional logic is trivial because propositional validity can always be decided mechanically in finitely many steps. In terms of computational complexity, however, propositional logic is by no means trivial because any of the known decision algorithms for propositional validity in the worst case is exponential in the number of "elementary" parts of the given formulae. In fact, the problem of propositional validity is one of the "NP-complete" problems all of which are considered to be computationally "hard".

For a formal treatment of these ideas, we first define "propositional formulae" and validity of propositional formulae. Then we define the "propositional transforms"  $\hat{p}$  of predicate logical formulae  $p$  and show that if  $\hat{p}$  is valid in propositional logic then  $p$  is valid in predicate logic.

**Definition 2.52. (Propositional Variables)**  $V$  is a set of *proposition logical* (or, just, propositional) *variables* iff  $V$  is a set of variables.

From the context it will always be clear when certain variables are used as predicate logical or proposition logical variables.

**Definition 2.53. (Propositional Formulae)** The set of *proposition logical formulae* (or, just, propositional formulae) is the smallest set of strings that satisfies the following conditions:

If  $v$  is a propositional variable then  $v$  is a propositional formula.

If  $b$  is a propositional formula then  $\neg b$  is a propositional formula.

If  $b_1$  and  $b_2$  are propositional formulae  
then  $\vee b_1 b_2$ ,  $\wedge b_1 b_2$ ,  $\rightarrow b_1 b_2$ , and  $\leftrightarrow b_1 b_2$   
are propositional formulae.

As in the case of predicate logical formulae, we will allow the use of parentheses and infix notation of propositional connectives in the "extended" proposition logical language.

We use the typed variables  $b, c$  as variables for proposition logical formulae.

Let now a fixed set  $V$  of propositional variables be given.



**Definition 2.54. (Truth Valuation)**

$T$  is a *truth valuation* iff

$T$  is a mapping from  $V$  into the set  $\{\mathbf{T}, \mathbf{F}\}$  of truth values.

**Definition 2.55. (Truth Values under Truth Valuations)** For all propositional formulae  $b$  and truth valuations  $T$  we define  $\langle b \rangle_T$  (the “truth value of  $b$  under the truth valuation  $T$ ”) inductively as follows:

$\langle v \rangle_T = T(v)$  if  $v$  is a propositional variable.

$\langle \neg b \rangle_T = B_{\neg}(\langle b \rangle_T)$ .

$\langle \vee b_1 b_2 \rangle_T = B_{\vee}(\langle b_1 \rangle_T, \langle b_2 \rangle_T)$ .

$\langle \wedge b_1 b_2 \rangle_T = B_{\wedge}(\langle b_1 \rangle_T, \langle b_2 \rangle_T)$ .

$\langle \rightarrow b_1 b_2 \rangle_T = B_{\rightarrow}(\langle b_1 \rangle_T, \langle b_2 \rangle_T)$ .

$\langle \leftrightarrow b_1 b_2 \rangle_T = B_{\leftrightarrow}(\langle b_1 \rangle_T, \langle b_2 \rangle_T)$ .

**Example 2.56. (Truth Values Under Truth Valuations)** Let  $b := \vee v \neg v$ , and let  $T$  be a truth valuation such that  $T(v) = \mathbf{T}$ . Then

$$\langle b \rangle_T = B_{\vee}(T(v), B_{\neg}(T(v))) = B_{\vee}(\mathbf{T}, B_{\neg}(\mathbf{T})) = \mathbf{T}.$$

**Definition 2.57. (Propositional Tautologies)**

$b$  is *valid in proposition logic* (or  $b$  is a *propositional tautology*) iff, for all truth valuations  $T$ ,  $\langle b \rangle_T = \mathbf{T}$ .

**Example 2.58. (Propositional Tautologies)** Each formula of the form  $\vee b \neg b$  is a propositional tautology because  $\langle \vee b \neg b \rangle_T = B_{\vee}(\langle b \rangle_T, B_{\neg}(\langle b \rangle_T))$  and therefore, for any truth valuation  $T$ , either  $\langle b \rangle_T = \mathbf{T}$  and  $\langle \vee b \neg b \rangle_T = \mathbf{T}$  or  $\langle b \rangle_T = \mathbf{F}$  and, again,  $\langle \vee b \neg b \rangle_T = \mathbf{T}$ .

**Definition 2.59. (Propositional Consequence)**

$b$  is a *propositional consequence* of  $c_1, \dots, c_n$  iff

for all truth valuations  $T$

if  $\langle c_1 \rangle_T = \dots \langle c_n \rangle_T = \mathbf{T}$  then  $\langle b \rangle_T = \mathbf{T}$ .

**Example 2.60. (Propositional Consequence)** Let  $b$  and  $c$  be any propositional formulae. Then  $c$  is a proposition logical consequence of  $b$  and  $b \rightarrow c$  because for any truth valuation  $T$  for which  $\langle b \rangle_T = \langle b \rightarrow c \rangle_T = \mathbf{T}$ , also  $\langle c \rangle_T$  must be  $\mathbf{T}$ .

**Exercise 2.61. (Characterization of Propositional Consequence)** Show that

$b$  is a propositional consequence of  $c_1, \dots, c_n$

iff

$((c_1 \wedge \dots \wedge c_n) \rightarrow b)$  is a propositional tautology.

*Solution:* : The proof is similar to the proof shown in Exercise 2.44:

$b$  is a propositional consequence of  $c_1, \dots, c_n$   
iff  
for all truth valuations  $T$ ,  
if  $\langle c_1 \rangle_T = \dots = \langle c_n \rangle_T = \mathbf{T}$  then  $\langle b \rangle_T = \mathbf{T}$   
iff  
for all truth valuations  $T$ ,  
 $B_{\rightarrow}(B_{\wedge}(\langle c_1 \rangle_T, \dots, \langle c_n \rangle_T), \langle b \rangle_T) = \mathbf{T}$   
iff  
for all truth valuations  $T$ ,  
 $\langle (c_1 \wedge \dots \wedge c_n) \rightarrow b \rangle_T = \mathbf{T}$   
iff  
 $(c_1 \wedge \dots \wedge c_n) \rightarrow b$  is a propositional tautology.

**Definition 2.62. (Elementary Formulae in Predicate Logic)**

$p$  is an *instantiation* (or *existential quantifier formula*) iff  
 $p$  has the form  $\exists v q$ .

$p$  is an *elementary formula* iff  
 $p$  is an atomic formula or  $p$  is an instantiation.

For each predicate logical formula  $p$  we now define the “propositional transform”  $\hat{p}$ . In fact, we only present an informal definition and leave an inductive definition as an exercise to the reader. In the examples we take  $w_1, w_2, \dots$  as the propositional variables.

**Definition 2.63. (Propositional Transforms of Formulae)** A propositional transform  $\hat{p}$  of a predicate logical formula  $p$  is a propositional formula that results from  $p$  by replacing the distinct elementary subformulae of  $p$  that are not themselves subformulae of elementary subformulae of  $p$  by distinct propositional variables.

**Example 2.64. (Propositional Transforms of Formulae)** Let  $p := ((\exists x (x > 0)) \vee (\exists y (y < 0))) \rightarrow ((x > 0) \wedge (\exists x (x > 0)))$ . Then  $\hat{p} = (w_1 \vee w_2) \rightarrow (w_3 \wedge w_1)$ .

**Theorem 2.65. (Proposition and Predicate Logical Consequences)**

If  $\hat{p}$  is a propositional tautology  
then  $p$  is a predicate logical tautology.

If  $\hat{p}$  is a propositional consequence of  $\hat{q}_1, \dots, \hat{q}_n$   
then  $p$  is a predicate logical consequence of  $q_1, \dots, q_n$ .

**Proof:**

*Tautology:* Let  $\hat{p}$  be a propositional tautology and  $I$  an interpretation in a domain  $\Delta$ . For simplicity of notation let us assume that  $p$  contains exactly one free variable  $v$  and let  $\gamma \in \Gamma$ . Let  $e_1, \dots, e_m$  be the distinct “maximal”

elementary subformulae of  $p$ . Then also  $(e_1)_v[\gamma'], \dots, (e_m)_v[\gamma']$  are elementary formulae from which  $p_v[\gamma']$  is built by propositional connectives exactly in the same way as  $\hat{p}$  is built from  $\hat{e}_1, \dots, \hat{e}_m$ . Hence, since  $\hat{p}$  is a propositional tautology, whatever the truth values  $\langle (e_1)_v[\gamma'] \rangle_I, \dots, \langle (e_m)_v[\gamma'] \rangle_I$  may be,  $\langle p_v[\gamma'] \rangle_I$  will be T.

*Consequence:* Let  $I$  be an interpretation in a domain  $\Delta$  such that  $q_1, \dots, q_n$  are valid under  $I$  in  $\Delta$ . Since  $\hat{p}$  is a propositional consequence of  $\hat{q}_1, \dots, \hat{q}_n$ ,  $((\hat{q}_1 \wedge \dots \wedge \hat{q}_n) \rightarrow \hat{p})$  is a propositional tautology. Hence,  $((q_1 \wedge \dots \wedge q_n) \rightarrow p)$  is also a predicate logical tautology. Hence,  $p$  is valid under  $I$  in  $\Delta$  by semantical modus ponens.  $\square$

In view of the above theorem we also define

**Definition 2.66. (Propositional Consequence for Predicate Logic Formulae)** Let  $p, q_1, \dots, q_n$  be predicate logical formulae.

$p$  is a propositional consequence of  $q_1, \dots, q_n$  iff  
 $\hat{p}$  is a propositional consequence of  $\hat{q}_1, \dots, \hat{q}_n$ .

**Definition 2.67. (Satisfiability)**

$b$  is satisfiable in proposition logic iff,  
 for some truth valuation  $T$ ,  $\langle b \rangle_T = T$ .

A set  $B$  of propositional formulae is satisfiable in proposition logic iff,  
 for some truth valuation  $T$ ,  
 for all  $b \in B$ ,  $\langle b \rangle_T = T$ .

**Exercise 2.68. (Tautologies, Consequences and Satisfiability)** Show that

$b$  is satisfiable in proposition logic  
 iff  
 $\neg b$  is not a proposition logical tautology.

$\langle a \rangle$

Hence, also

$b$  is a propositional tautology  
 iff  
 $\neg b$  is not satisfiable in proposition logic

$\langle a \rangle$

and

~~$b$  is a propositional consequence of  $c_1, \dots, c_n$   
 iff  $c_1 \wedge \dots \wedge c_n \wedge \neg b$  is not satisfiable.~~

$\langle \text{not} \rangle$

*Solution:*

$b$  is satisfiable in proposition logic  
 iff  
 for some truth valuation  $T$ ,  $\langle b \rangle_T = T$

$b$  taut.  $\Leftrightarrow$   
 $\neg b$  not taut.  $\Leftrightarrow$   
 $\neg b$  not sat.  $\Leftrightarrow$   
 $b$  satisfiable

iff  
 for some truth valuation  $T$ ,  $\langle \neg b \rangle_T = F$ .  
 iff  
 not for all truth valuations  $T$ ,  $\langle \neg b \rangle_T = T$   
 iff  
 $\neg b$  is not a propositional tautology.

**Exercise 2.69. (Proposition Logical Consequence and Insatisfiability)** Show that

$b$  is a propositional consequence of  $c_1, \dots, c_n$  iff  
 $(c_1 \wedge \dots \wedge c_n \wedge \neg b)$  is not satisfiable in propositional logic.

*Solution:*

$b$  is a propositional consequence of  $c_1, \dots, c_n$   
 iff  
 $((c_1 \wedge \dots \wedge c_n) \rightarrow b)$  is a propositional tautology  
 iff  
 $\neg((c_1 \wedge \dots \wedge c_n) \rightarrow b)$   
 is not satisfiable in propositional logic  
 iff  
 $(c_1 \wedge \dots \wedge c_n \wedge \neg b)$  is not satisfiable in propositional logic.

**Exercise 2.70. (Checking Proposition Logical Consequences)** For any formulae  $p_1, p_2, p_3$ , show that  $p_3$  is a propositional consequence of  $p_2$  and  $((p_1 \vee p_2) \rightarrow p_3)$ .

One method for showing this is by using the lemma on propositional consequence and tautology. Namely, we calculate the following "truth table".

$v_1$	$v_2$	$v_3$	$B_{\rightarrow}(B_{\wedge}(v_2, B_{\rightarrow}(B_{\vee}(v_1, v_2), v_3)), v_3)$
T	T	T	T
T	T	F	T
T	F	T	T
T	F	F	T
F	T	T	T
F	T	F	T
F	F	T	T
F	F	F	T

Here, for  $i = 1, 2, 3$ ,  $v_i = \langle \hat{p}_i \rangle_T$  (for an arbitrary but fixed truth valuation  $T$ ) and hence  $B_{\rightarrow}(B_{\wedge}(v_2, B_{\rightarrow}(B_{\vee}(v_1, v_2), v_3)), v_3) = \langle (\hat{p}_2 \wedge ((\hat{p}_1 \vee \hat{p}_2) \rightarrow \hat{p}_3)) \rightarrow \hat{p}_3 \rangle_T$ .

The method of "truth tables" needs  $2^n$  steps when  $n$  is the number of elementary formulae in the given formulae.

Another method for showing propositional consequences is the "indirect" method which is sometimes faster. It proceeds by using the lemma on propositional consequence and insatisfiability. In our example, we assume that for some  $T$

*probably because many formulae contain several "→" in the same direction*



We will now present an alternative method for deciding whether or not a propositional formula  $b$  is a propositional tautology and whether  $b$  is a propositional consequence of  $c_1, \dots, c_n$ . This method is called "resolution method". Its predicate logic variant plays an important role in automated theorem proving and, in particular, in logic programming. Predicate logic resolution is based on two ideas:

*only has  
and a  
and  
predicate  
ification  
in principle.*

1. repeated resolution of "complementary literals" and
2. "unification".

The idea of resolution of complementary literals can already be studied in propositional logic.

The propositional resolution method works only for formulae in "clause form" (conjunctive normal form), also called "clause sets". Therefore, before applying the method to an arbitrary propositional formula, we must transform the formula into an equivalent formula in clause form. In the sequel, we present the details of both the transformation into clause form and repeated resolution of complementary literals.

**Definition 2.71. (Propositional Clause Sets)**

$K$  is a *propositional clause set* iff  
 $K$  is a finite set of propositional clauses.

$C$  is a *propositional clause* iff  
 $C$  is a finite set of propositional literals.

$l$  is a *propositional literal* iff  
 $l$  has the form  $v$  or the form  $\neg v$ ,  
 where  $v$  is a propositional variable.

**Example 2.72. (Propositional Formulae in Clause Form)** The following set  $K$  is a propositional clause set:

$$\begin{aligned} K &= \{C_1, C_2, C_3\}, \text{ where} \\ C_1 &= \{w_1, w_2\}, \\ C_2 &= \{\neg w_2\}, \\ C_3 &= \{\neg w_1, w_2, \neg w_3\}. \end{aligned}$$

□

In fact, propositional clause sets and clauses are compact descriptions of certain propositional formulae as defined by the following definition. (For " $K^*$ " etc. read "the propositional formula determined by  $K$ " etc. Note, that strictly speaking the formula described by a clause set or a clause are not uniquely determined by the subsequent definition because the order of the elements in sets is not unique. We could overcome this ambiguity by introducing some order on propositional variables but we do not want to overformalize the subject at this point.)

**Definition 2.73. (The Formula Determined by a Clause Set)**

If  $C = \{l_1, \dots, l_m\}$  is a propositional clause  
 then  $C^* = l_1 \vee \dots \vee l_m$ .

If  $K = \{C_1, \dots, C_n\}$  is a propositional clause set  
 then  $K^* = C_1^* \wedge \dots \wedge C_n^*$ .

*( $K^*$  is in clause form, i.e. in conjunctive normal form, a conj. of disj. of literals)*

**Example 2.74. (The Formula Determined by a Clause Set)** Let  $K, C_1, C_2, C_3$  be as in Example 2.78. Then

$$K^* = C_1^* \wedge C_2^* \wedge C_3^*, \text{ where}$$

$$C_1^* = w_1 \vee w_2,$$

$$C_2^* = \neg w_2,$$

$$C_3^* = \neg w_1 \vee w_2 \vee \neg w_3. \quad \square$$

The reason why, in the context of resolution, clauses and conjunctions of clauses are written in set form is the fact that we want to consider two clauses to be "equal" if they contain the same literals but maybe in different order and/or with multiple occurrences. Similarly, we want to identify conjunctions of clauses that contain the same clauses but in different order and/or multiple occurrences.

There exists an algorithm by which every propositional formula can be transformed into an "equivalent" formula that can be described by a clause set.

**Definition 2.75. (Equivalence of Propositional Formulae)**

$b \sim c$  (" $b$  is propositionally equivalent to  $c$ ") iff  
 for all truth valuations  $T, \langle b \rangle_T = \langle c \rangle_T$ .

**Proposition 2.76. (Equivalences for Clause Form Transformation)**

$$b \leftrightarrow c \sim (b \rightarrow c) \wedge (c \rightarrow b).$$

$$b \rightarrow c \sim \neg b \vee c.$$

$$\neg \neg b \sim b.$$

$$\neg(b_1 \vee b_2) \sim (\neg b_1 \wedge \neg b_2).$$

$$\neg(b_1 \wedge b_2) \sim (\neg b_1 \vee \neg b_2).$$

$$b \vee (c_1 \wedge c_2) \sim ((b \vee c_1) \wedge (b \vee c_2)).$$

$$(c_1 \wedge c_2) \vee b \sim ((c_1 \vee b) \wedge (c_2 \vee b)).$$

$$b \vee b \sim b.$$

$$(b_1 \vee b_2) \vee b_3 \sim b_1 \vee (b_2 \vee b_3).$$

*These formulae, read from left to right, transform formulae into CNF (= clause form).*

*A formula is said to be in conjunctive normal form (CNF) iff it is a "fundamental conjunction":*

$$(\dots) \wedge (\dots) \wedge (\dots) \wedge \text{etc}$$

*no conjunctions inside (i.e. only disjunctions)*

$$b_1 \vee b_2 \sim b_2 \vee b_1.$$

$$b \wedge b \sim b.$$

$$(b_1 \wedge b_2) \wedge b_3 \sim b_1 \wedge (b_2 \wedge b_3).$$

$$b_1 \wedge b_2 \sim b_2 \wedge b_1.$$

**Proof:** : Exercise.

**Proposition 2.77. (Transformation into Clause Form)** There exists an algorithm by which for every propositional formula  $b$  one can find a clause set  $K$  such that  $b$  is equivalent to  $K^*$ .

**Proof:** By "applying" the equivalences stated in the preceding proposition we can, first, transform  $b$  into an equivalent formula  $c$  that is a conjunction of disjunctions of literals (a formula in "conjunctive normal form") with the additional property that all disjunctions contain a literal only once and that no two disjunctions occur whose set of literals is identical. Every disjunction of  $c$  can then be described by a set of literals. The set of these sets is then the desired  $K$ .

**Example 2.78. (Transformation into Clause Form)** Let  $b := (w_1 \vee w_2) \rightarrow (w_3 \vee w_3)$ . The following equivalence steps yield an equivalent formula  $c$  in conjunctive normal form.

$$\begin{aligned} &(w_1 \vee w_2) \rightarrow (w_3 \vee w_3) \\ &\sim \\ &(w_1 \vee w_2) \rightarrow w_3 \\ &\sim \\ &\neg(w_1 \vee w_2) \vee w_3 \\ &\sim \\ &(\neg w_1 \wedge \neg w_2) \vee w_3 \\ &\sim \\ &(\neg w_1 \vee w_3) \wedge (\neg w_2 \vee w_3) =: c. \end{aligned}$$

*CNF!*

The corresponding clause set  $K$  is

$$\{\{\neg w_1, w_3\}, \{\neg w_2, w_3\}\}.$$

*In a bit more general formulation (the method), this is only inference rule sufficient for any proof in mathematics. (Unit clause corresponds to the first step in the proof)*

$$\left\{ \begin{array}{l} 3x + 7y \\ 2x - y \end{array} \right\} \xrightarrow{\text{LWR}} \left\{ \begin{array}{l} 3x + 7y \\ 4x - 2y \end{array} \right\} \xrightarrow{\text{RES}} \left\{ \begin{array}{l} 3x + 7y \\ 4x - 2y \end{array} \right\} \rightarrow 7x \quad \square$$

Now we proceed to proving an elementary lemma that will be the basis of the propositional resolution method.

**Lemma 2.79. (Correctness of a Propositional Resolution Step)**

$b_1 \vee b_2$  is a propositional consequence of  $c \vee b_1$  and  $\neg c \vee b_2$ .

*but not equivalence (just as in arithmetic!)*

	<i>resolution</i>	<i>step elimination</i>
	<i>logic:</i>	<i>arithmetic:</i>
	$x \vee y$	$x + y$
	$x \vee \neg y$	$x - y$
	$x$	$x$

**Proof:** : Let  $T$  be an arbitrary truth valuation and assume that



*Handwritten proof:*

$$\left[ \begin{aligned} (c \vee b_1) \wedge (\neg c \vee b_2) &\rightarrow b_1 \wedge b_2 \sim (c \wedge \neg c) \vee (c \wedge b_2) \vee (\neg c \wedge b_1) \\ &\sim [(c \wedge \neg c) \vee (c \wedge b_2) \vee (\neg c \wedge b_1)] \wedge [(c \wedge \neg c) \vee (c \wedge b_2) \vee (\neg c \wedge b_1)] = \mathbf{T} \end{aligned} \right]$$

Contents 65

$$\langle c \vee b_1 \rangle_T = \mathbf{T} \text{ and} \tag{1}$$

$$\langle \neg c \vee b_2 \rangle_T = \mathbf{T}. \tag{2}$$

If  $\langle b_1 \rangle_T = \mathbf{T}$  then, of course,  $\langle b_1 \vee b_2 \rangle_T = \mathbf{T}$ . If  $\langle b_1 \rangle_T = \mathbf{F}$  then  $\langle c \rangle_T = \mathbf{T}$  by (1) and, hence,  $\langle \neg c \rangle_T = \mathbf{F}$ . Therefore,  $\langle b_2 \rangle_T = \mathbf{T}$  by (2) and, hence,  $\langle b_1 \vee b_2 \rangle_T = \mathbf{T}$ .  $\square$ .

We now define one basic step in the resolution method, namely formation of "resolvents". For this we need the auxiliary notion of "complementary literals".

**Definition 2.80. (Complementary Literals)**

Two propositional literals  $l'$  and  $l''$  are *complementary* iff  
 $l' = v$  and  $l'' = \neg v$  (or viceversa),  
 where  $v$  is a propositional variable. *i.e. (a.v) ( $l' = v \wedge l'' = \neg v$ ) / ( $l' = \neg v \wedge l'' = v$ )*  
*with level*

**Definition 2.81. (Propositional Resolvents)** Let  $C' = \{l'_1, l'_2, \dots, l'_m\}$  and  $C'' = \{l''_1, l''_2, \dots, l''_n\}$  be two propositional clauses with the property that  $l'_1$  and  $l''_1$  are complementary literals. Then

$C$  is a *propositional resolvent* of  $C'$  and  $C''$  iff  
 $C = \{l'_2, \dots, l'_m, l''_2, \dots, l''_n\}$ .

**Example 2.82. (A Propositional Resolvent)**  $\{w_2, w_3\}$  is a propositional resolvent (and in fact the only possible resolvent) of  $\{w_1, w_2\}$  and  $\{\neg w_1, w_3\}$ .  $\square$

Iteration of resolvent formation yields the resolution method:

**Theorem 2.83. (Propositional Resolution Method)** Consider the following problem:

Given:  $K$ , a propositional clause set.  
 Question: Is  $K^*$  satisfiable?

$$\left[ \begin{aligned} B \text{ Tautology} &\Leftrightarrow \neg B \text{ unsatisfiable} \text{ !} \\ [A_1, \dots, A_n \vdash B] &\Leftrightarrow (A_1 \wedge \dots \wedge A_n \rightarrow B) \end{aligned} \right]$$

This problem can be solved by the following algorithm:

```

K' := K
while exists C such that
  C is a propositional resolvent of two clauses in K' and
  C ∉ K' (identical equations are not written twice)
do
  if C = ∅ / so-called "empty clause" □
  then answer: "Not satisfiable"
  else K' := K' ∪ {C}
answer: "Satisfiable".
  
```

**Proof:** We have to show

1. The algorithm always terminates. *for propositional logic (and for predicate logic)*
2. If the algorithm terminates with the answer "not satisfiable" then  $K$  is not satisfiable.

3. If the algorithm terminates with the answer "satisfiable" then  $K$  is satisfiable.

*Termination:* The algorithm always terminates because from the finitely many literals occurring in the initial clause set  $K$  one can form only finitely many distinct clauses. (If in  $K$  there occur  $n$  distinct variables then only  $3^n$  many literals can be formed.)

*Answer "not satisfiable":* The answer "not satisfiable" can only be produced if the clause  $\emptyset$  is generated as a resolvent. The empty clause can only be generated from two clauses of the form  $\{l\}$  and  $\{l'\}$ , where the literals  $l$  and  $l'$  are complementary. By the lemma on the correctness of the propositional resolution step, each clause produced in the course of the above algorithm is a propositional consequence of  $K$ . Hence,  $l$  and  $l'$  are propositional consequences of  $K$ . Hence, if a truth valuation  $T$  satisfied  $K$ , it would satisfy both  $l$  and  $l'$ , which of course is not possible since  $l$  and  $l'$  are complementary.

*Answer "satisfiable":* This proof is not given here. (It is included in the later proof of the correctness of the predicate logic resolution method.) The proof consists in showing how, if the answer is "satisfiable", one can construct a satisfying truth valuation for  $K$  from the final clause set  $K'$ . We illustrate the construction in the Example 2.85 below.  $\square$

The resolution method is only applicable to formulae in clause form. In order to decide whether arbitrary propositional formulae are tautologies or whether a propositional formula is a logical consequences of other propositional formulae one can apply the lemmata stated in Exercise 2.68 and transform the resulting formulae into clause form by Proposition 2.77

**Example 2.84. (Resolution, Unsatisfiable Case)** Decide by the resolution method whether the following clause set is satisfiable:

- (1)  $w_1 \vee w_2,$
- (2)  $w_1 \vee \neg w_3,$
- (3)  $\neg w_1 \vee w_3,$
- (4)  $\neg w_1 \vee \neg w_2,$
- (5)  $w_3 \vee \neg w_2,$
- (6)  $\neg w_3 \vee w_2,$

(Note that, in examples, we often write clause sets as a sequence of clauses that are presented as disjunctions. However, this notation should be understood as a notation for sets.)

*Solution:*

- (7)  $w_2 \vee w_3,$  (from (1) and (3))
- (8)  $w_2,$  (from (6) and (7))
- (9)  $\neg w_3 \vee \neg w_2,$  (from (2) and (4))
- (10)  $\neg w_2,$  (from (5) and (9))
- (11)  $\emptyset.$  (from (8) and (10))

(11)  $\emptyset$  from (8) and (10) - not  $\exists!$

~~3~~

$(v_1, \dots$

$\left\{ \begin{array}{l} v_1 \text{ app} \\ v_2 \text{ app} \\ v_3 \text{ app} \end{array} \right.$

$\left[ \begin{array}{l} v_1, v_2 \text{ app} \\ \rightarrow \text{ can} \end{array} \right.$

The system is "saturated" "stable".

Answer: "Not satisfiable".

**Example 2.85. (Resolution, Satisfiable Case)** Decide by the resolution method whether the following clause set is satisfiable and, if the answer is "satisfiable", construct a satisfying truth valuation for the clause set.

- (1)  $w_1 \vee w_2$ ,
- (2)  $\neg w_2$ ,
- (3)  $\neg w_1 \vee w_2 \vee \neg w_3$ .

*Solution:*

- (4)  $w_1$ , (from (1) and (2))
- (5)  $w_2 \vee \neg w_3$ , (from (1) and (3))
- (6)  $\neg w_1 \vee \neg w_3$ , (from (2) and (3))
- (7)  $\neg w_3$ . (from (2) and (5))

At this stage, no new resolvent can be generated.

- (1) with (4): There is no complementary literal.
- (1) with (5): There is no complementary literal.
- (1) with (6):  $w_2 \vee \neg w_3$  is a resolvent. However, it is not new.
- ...
- (6) with (7): There is no complementary literal.

*not always so clear  
readable one from  
the more resolution*

Answer: "Satisfiable".

A satisfying truth valuation  $T$  can be generated as follows:

- $T(w_1) = \mathbf{T}$ , (from (4))
- $T(w_2) = \mathbf{F}$ , (from (2))
- $T(w_3) = \mathbf{F}$ . (from (7))

It is easily seen that this truth valuation satisfies all clauses (1) - (7), which is in fact guaranteed by theory.

**Example 2.86. (Resolution; Logical Consequences)** Decide by the resolution method whether

$$b := \neg w_1 \wedge \neg w_2$$

is a propositional consequence of

$$c_1 := (w_1 \vee w_2) \rightarrow w_3 \text{ and}$$

$$c_2 := \neg w_3.$$

*Solution:*

First we transform the problem into a satisfiability problem of a clause set:  $b$  is a propositional consequence of  $c_1$  and  $c_2$  iff the formula  $d := c_1 \wedge c_2 \wedge \neg b$  is not satisfiable. In order to apply resolution we first transform  $d$  into clausal form. Note that in this case (and in fact in all cases where the formula considered derives from a propositional consequence problem), transformation into clausal form can be done separately for each part of the conjunction, which considerably reduces the work involved.

By Example 2.78 the clausal form of  $c_1$  is

- (1)  $\neg w_1 \vee w_3,$   
 (2)  $\neg w_2 \vee w_3.$

$c_2$  is already in clause form. Hence,

- (3)  $\neg w_3.$

The clausal form of  $\neg b$  is

- (4)  $w_1 \vee w_2,$

because

$$\neg b \sim \neg(\neg w_1 \vee \neg w_2) \sim w_1 \vee w_2.$$

Now we can apply resolution to the clauses (1) - (4):

- |                  |                    |
|------------------|--------------------|
| (5) $\neg w_1,$  | (from (1) and (3)) |
| (6) $\neg w_2,$  | (from (2) and (3)) |
| (7) $w_2,$       | (from (4) and (5)) |
| (8) $\emptyset.$ | (from (6) and (7)) |

Answer: "Unsatisfiable", i.e.  $b$  is a propositional consequence of  $c_1$  and  $c_2$ .  $\square$

Propositional formulae "describe boolean functions". We will now prove that, conversely, all boolean functions can be described by propositional formulae. In fact, all boolean functions can be described by propositional formulae involving only  $\neg$  and  $\vee$  because every propositional formula is equivalent to a propositional formula involving only  $\neg$  and  $\vee$ . The set  $\{\neg, \vee\}$  is therefore called a "complete set of propositional operators". Another complete sets of operators are  $\{\neg, \wedge\}$  because  $b_1 \vee b_2$  is equivalent to  $\neg(\neg b_1 \wedge \neg b_2)$ . A third set of complete operators is  $\{\mid\}$ , where  $b_1 \mid b_2$  is defined to be equivalent to  $\neg(b_1 \wedge b_2)$ . " $\mid$ " is sometimes called "nand". The set  $\{\mid\}$  is complete because  $\neg b$  is equivalent to  $b \mid b$  and  $b_1 \wedge b_2$  is equivalent to  $\neg(b_1 \mid b_2)$  and, therefore, to  $(b_1 \mid b_2) \mid (b_1 \mid b_2)$ . *Yet another complete set:  $\{\vee\}$  with  $b_1 \vee b_2 := \neg(b_1 \mid b_2)$  (conv.).*

For the following study, let  $V = \{w_1, w_2, \dots\}$  be fixed. Let  $w_1, w_2, \dots$  range over  $\{T, F\}$ .

#### Definition 2.87. (Boolean Functions Described By Formulae)

$B$  is an  $n$ -ary boolean function iff  $B: \{T, F\}^n \rightarrow \{T, F\}$ .

Let all variables of the propositional formula  $b$  be among the variables  $w_1, w_2, \dots, w_n$ . Then:

$[b]_n$  ("the  $n$ -ary boolean function described by  $b$ ") is defined as follows:

$$[b]_n(w_1, \dots, w_n) = \langle b \rangle_T,$$

where  $T(w_i) := w_i$  for all  $1 \leq i \leq n$ .

Lemma 2.88. (Complete Sets of Boolean Operators) Let  $n \geq 1$ .

For every  $n$ -ary boolean function  $B$   
 there exists a propositional formula  $b$  (involving only  $\neg$  and  $\vee$ )  
 such that  $B = \langle b \rangle_n$ .

$\boxed{\langle b \rangle}_n$

**Proof:** We give two proofs that are seemingly different but, in fact, are only two versions, one recursive and one non-recursive, of the same idea. Both proofs show that all  $n$ -ary boolean functions can be described by propositional formulae involving only  $\neg$ ,  $\vee$ , and  $\wedge$ . The result then follows from the fact that  $b_1 \wedge b_2$  is equivalent to  $\neg(\neg b_1 \vee \neg b_2)$ .

*Inductive Proof.*

The possible four unary boolean functions can be described by the propositional formulae  $w_1$ ,  $\neg w_1$ ,  $w_1 \wedge \neg w_1$ , and  $w_1 \vee \neg w_1$ . (This is the "induction start".)

As induction hypothesis, let us assume that all  $(n-1)$ -ary boolean functions can be described by propositional formulae involving only  $\neg$ ,  $\vee$  and  $\wedge$ . Let now  $n > 1$  and let  $B$  be an  $n$ -ary boolean function.

It is easy to check that

$$B(w_1, \dots, w_n) = B_{\vee}(B_{\wedge}(w_1, B(\mathbf{T}, w_2, \dots, w_n)), B_{\wedge}(B_{\neg}(w_1), B(\mathbf{F}, w_2, \dots, w_n))). \quad (1)$$

(Hint: consider the two cases  $w_1 = \mathbf{T}$  and  $w_1 = \mathbf{F}$ .) Now,

$$b_{2, \dots, w_n} := \lambda w_2, \dots, w_n (B(\mathbf{T}, w_2, \dots, w_n)) \text{ and}$$

$$c_{2, \dots, w_n} := \lambda w_2, \dots, w_n (B(\mathbf{F}, w_2, \dots, w_n))$$

are  $(n-1)$ -ary boolean functions and therefore, by induction hypothesis, can be described by two propositional formulae  $b_1$  and  $b_2$ . Obtain  $b'_1$  and  $b'_2$  from  $b_1$  and  $b_2$ , respectively, by simultaneously replacing  $w_1$  by  $w_2, \dots, w_{n-1}$  by  $w_n$ . Then, using (1),  $B$  can be described by

$$(w_1 \wedge b'_1) \vee (\neg w_1 \wedge b'_2).$$

$\uparrow$  (replace  $w_1$  with  $w_2, \dots, w_{n-1}$  etc.)

*Iterative Proof.*

Let  $B$  be an  $n$ -ary boolean function and let  $(w_1^{(1)}, \dots, w_n^{(1)}), (w_1^{(2)}, \dots, w_n^{(2)}), \dots, (w_1^{(l)}, \dots, w_n^{(l)})$  be the distinct  $n$ -tuples in  $\{\mathbf{T}, \mathbf{F}\}^n$  for which  $B$  has the value  $\mathbf{T}$ . Then  $B$  can be described by the following propositional formula (in DNF)

$$b = c_{(w_1^{(1)}, \dots, w_n^{(1)})} \vee c_{(w_1^{(2)}, \dots, w_n^{(2)})} \vee \dots \vee c_{(w_1^{(l)}, \dots, w_n^{(l)})},$$

where

$$c_{(w_1, \dots, w_n)} = w_1^{w_1} \wedge \dots \wedge w_n^{w_n}$$

$$\text{and } w_i^w = \begin{cases} w_i & \text{if } w = \mathbf{T} \\ \neg w_i & \text{otherwise.} \end{cases}$$

This can easily be checked. (Hint: Consider the two cases  $B(w_1, \dots, w_n) = \mathbf{T}$  and  $B(w_1, \dots, w_n) = \mathbf{F}$  and compute  $\langle b \rangle_n(w_1, \dots, w_n)$  by tracing the definition of  $b$ .)

**Example 2.89. (Description of a Boolean Function)** Consider the boolean ternary function defined by the following table:

$w_1$	$w_2$	$w_3$	$B(w_1, w_2, w_3)$
T	T	T	T
T	T	F	T
T	F	T	F
T	F	F	F
F	T	T	T
F	T	F	F
F	F	T	F
F	F	F	T

In this example,

$$M := \{(T, T, T), (T, T, F), (F, T, T), (F, F, F)\}.$$

and therefore

$$b := (w_1 \wedge w_2 \wedge w_3) \vee (w_1 \wedge w_2 \wedge \neg w_3) \vee (\neg w_1 \wedge w_2 \wedge w_3) \vee (\neg w_1 \wedge \neg w_2 \wedge \neg w_3).$$

### 2.5.7 A Lemma on Semantics and Substitution

For some of the proofs in the subsequent chapters one needs a technical lemma that relates substitution with semantics. Roughly, this lemma shows that the meaning of the formula  $p_v[t]$  is identical to the meaning of the formula  $p_v[\gamma']$ , where  $\gamma'$  is the name of the object  $\gamma$  that is the meaning of  $t$ .

**Lemma 2.90. (Semantics and Substitution)** Let  $\Delta$  be a domain,  $I$  an interpretation,  $t$  a variable-free term,  $s$  a term that contains at most the one variable  $v$  and  $p$  a formula that has at most the one free variable  $v$ . Then

$$\langle s_v[t] \rangle_I = \langle s_v[\gamma'] \rangle_I \text{ and}$$

$$\langle p_v[t] \rangle_I = \langle p_v[\gamma'] \rangle_I,$$

where  $\gamma := \langle t \rangle_I$ .

**Proof:** The proof of this lemma is tedious but not difficult. It is inductive in nature (induction over the length of formulae) and follows the pattern of many of the previous proofs on properties of substitution etc.

[ *The details of this proof will be included in a later version of these lecture notes.* ]

**Example 2.91. (Semantics and Substitution)** Let

$$p := \exists y (x = 2 \cdot y)$$

$$v := x, \text{ and}$$

$$t := 5 \cdot 3 + 1.$$

Now, under the usual interpretation of “+” and “.” in the domain of natural numbers,

$$\langle 5 \cdot 3 + 1 \rangle_I = 16,$$

$$\langle \exists y (5 \cdot 3 + 1 = 2 \cdot y) \rangle_I = \mathbf{T}, \text{ and also}$$

$$\langle \exists y (16 = 2 \cdot y) \rangle_I = \mathbf{T}.$$

### 3. Reasoning in Predicate Logic

We have seen that, in general, it is not immediately possible to determine whether or not a formula  $p$  is a (predicate) logical consequence of a set of formulae  $Q$  because, following the definition of "logical consequence", one would have to observe whether  $p$  is valid under "all" possible interpretations under which all formulae of  $Q$  are valid. Even one single determination of the truth value  $\langle p \rangle_I$  may be a problem if  $p$  involves quantifiers because, by the definition of  $\langle p \rangle_I$  for formulae involving quantifiers, one would have to "observe" the truth value  $\langle q \rangle_I$  for infinitely many formulae  $q$ .

The objective of "reasoning" is to establish the validity of formulae by syntactical transformation of initial formulae, i. e. by the application of "rules of inference". Surprisingly, for first order predicate logic, it can be shown (Gödel's Completeness Theorem) that any logical consequence can be established by a finite number of applications of certain simple rules of inference. In fact, infinitely many such systems of inference rules could be established. (Compare: each algorithmically solvable problem can be solved by infinitely many different algorithms.) We will consider just one such system (which is of the "Hilbert type"). For theoretical purposes, it is advisable to make such a system as simple as possible because, in the analysis of the system, we will have to consider only a few cases. For practical purposes, however, i. e. when the system is used for doing real proofs, a proof system is only feasible if it has a certain complexity. Therefore, we pursue the following strategy: We first introduce a simple system (the "kernel" system) which is the object of subsequent theoretical studies. For example, we will study "correctness" of the system. Then, we will show how more complicated proof techniques can be reduced to the simple rules of the kernel system. These proof techniques form the "extended proof system" that can be used for practical purposes. By the reduction process, the theoretical properties studied for the kernel system carry over to the extended system, compare the remarks about the "kernel" language and the "extended" language of predicate logic.

#### 3.1 Axioms, Inference Rules, Proofs, Theorems

In this section we present one particular reasoning system for first order predicate logic and formulate a notion of "proof".

**Definition 3.1.** (Axioms and Inference Rules of Predicate Logic) Let  $S$  be a domain of symbols. The following formulae are called "*axioms of (first order) predicate logic*" (over  $S$ ):

*Substitution Axioms:*



All formulae (over  $S$ ) of the form  $p_v[t] \rightarrow \exists v p$ .  
*Identity Axioms:*

All formulae (over  $S$ ) of the form  $v = v$ .

*Equality Axioms:*

All formulae (over  $S$ ) of the form  
 $(v_1 = w_1 \wedge \dots \wedge v_n = w_n) \rightarrow f v_1 \dots v_n = f w_1 \dots w_n$ .

All formulae (over  $S$ ) of the form  
 $(v_1 = w_1 \wedge \dots \wedge v_n = w_n) \wedge r v_1 \dots v_n \rightarrow r w_1 \dots w_n$ .

The following rules are called "*elementary inference rules first order predicate logic*":

*Propositional Rule:*

If  $p$  is a propositional consequence of  $q_1, \dots, q_n$   
 then  $p$  can be derived from  $q_1, \dots, q_n$ .

*$\exists$ -Introduction Rule:*

If  $v$  is not free in  $q$   
 then  $(\exists v p) \rightarrow q$  can be derived from  $p \rightarrow q$ . □

Note that the notion of a predicate logical axiom depends on the domain  $S$ . This is important in some of the more subtle considerations in the sequel.

**Definition 3.2. (Proofs)** Let  $S$  be a domain of symbols and let  $Q$  be a set of formulae (over  $S$ ).

A finite sequence  $\bar{p}$  of formulae (over  $S$ ) is a *proof in the theory  $Q$*  (over  $S$ ) iff

for all  $1 \leq i \leq \text{length}(\bar{p})$ ,

$\bar{p}_i \in Q$  or

$\bar{p}_i$  is an axiom of predicate logic or,

for some  $j_1, \dots, j_m < i$ ,  $\bar{p}_i$  can be derived from  $\bar{p}_{j_1}, \dots, \bar{p}_{j_m}$   
 by application of an inference rule.

$Q \vdash_S p$  (" $p$  is *provable* from  $Q$ " or " $p$  is a *theorem* in the theory  $Q$ ") iff  
 there exists a proof  $\bar{p}$  in  $Q$  (over  $S$ ) such that  
 $p$  is the last formula of  $\bar{p}$ . □

If  $S$  is clear from the context or fixed then we often write " $\vdash$ " instead of " $\vdash_S$ ".

**Example 3.3. (A Proof)** We give an example that shows two things: It shows a proof and, also, it shows how more complicated proof steps can be replaced by

more elementary proof steps. This method of "reduction" will be used several times below in order to establish the soundness of the many proof techniques used in practical proving.

*$\forall$ -Introduction Rule:*

If  $v$  is not free in  $p$  and  
 $Q \vdash p \rightarrow q$   
 then  $Q \vdash p \rightarrow \forall v q$ .

Here is a sequence of formulae that is a proof of  $p \rightarrow \forall v q$  "in the theory"  $\{p \rightarrow q\}$ :

- |     |  |                                       |
|-----|--|---------------------------------------|
| (1) | $p \rightarrow q,$                       |                                       |
| (2) | $\neg q \rightarrow \neg p,$             | (from (1) by the propositional rule)  |
| (3) | $(\exists v \neg q) \rightarrow \neg p,$ | (from (2) by $\exists$ -introduction) |
| (4) | $p \rightarrow \neg \exists v \neg q.$   | (from (3) by the propositional rule)  |

The last formula is an abbreviation for  $p \rightarrow \forall v q$ .  $\square$

### 3.2 Correctness of the Reasoning System

Proving (reasoning, inferring) in the above technical sense has all the essential properties we listed in the introduction of this book. It is *abstract* in the sense that it only considers the syntactical structure of formulae and not their meaning. It is *verifiable* in the sense that each individual step of inference can be checked by a computer. (Note that this does not mean that a computer is supposed to be able to *find* a suitable sequence of steps for proving a desired formula). We will now show that the above reasoning system is also *correct* in the sense that a formula that is derived by this system from formulae that are valid under a given interpretation is again valid under the same interpretation. The above reasoning system is also *universal* in the sense that it is independent of which theory we consider. (Much later we will show that the above reasoning system is also *universal* in the sense that any logical consequence can be derived by a finite number of reasoning steps.)

**Lemma 3.4. (Correctness of Reasoning in Predicate Logic)**

If  $Q \vdash p$  then  $Q \models p$ .

**Proof:** : We first show the following two facts:

1. All axioms of predicate logic are valid in predicate logic.
2. The conclusion of each inference rule is a predicate logical consequence of the premises.

The proof of these facts is tedious but not difficult. The proofs are inductive in nature (induction over the length of formulae) and follow the pattern of many of the previous proofs on properties of substitution etc. At certain stages Lemma 2.90 on semantics and substitution is necessary.

[ *The details of these proofs will be included in a later version of these lecture notes.* ]

Now we use these facts for the proof of the lemma by induction over the length of proofs. For this, let  $n$  be fixed and assume that

for all proofs  $\bar{p}$  of length  $< n$  in  $Q$   
and for all formulae  $p$  in  $\bar{p}$  we have  $Q \models p$ .

Let  $\bar{q}$  be a proof in  $Q$  of length  $n$  and let  $q$  be the last formula of  $\bar{q}$ . Then there are three cases:

*Case*  $q \in Q$ : In this case,  $Q \models q$ .

*Case*  $q$  is an axiom of predicate logic: By the first fact above,  $q$  is valid and, hence,  $Q \models q$ .

*Case* for certain  $j_1, \dots, j_m < \text{length}(\bar{q})$  the formula  $q$  is derived from  $\bar{q}_{j_1}, \dots, \bar{q}_{j_m}$  by an inference rule: By the second fact above

$$\{\bar{q}_{j_1}, \dots, \bar{q}_{j_m}\} \models q. \quad (1)$$

By the induction hypothesis,

$$Q \models \bar{q}_{j_1}, \quad (2.1)$$

...

$$Q \models \bar{q}_{j_m}, \quad (2.m)$$

From (1) and (2) we obtain  $Q \models q$ .

*Chin Peng*

## Basic Proof Techniques

### Proof Situations

Proving is stepwise arranging of "proof situations". A proof situation is characterized by the current "knowledge base" (set of sentences that are assumed to be true or are already proven and, hence, can be "used") and a sentence that should be proven. The goal of proving is to arrive at a point where all proof situations considered are "trivial". A proof situation is trivial if the sentence to be proven occurs in the knowledge base.

In the following subsections we compile the basic proof techniques by which the possible proof situations can be handled. The description of these techniques is informal. The proof techniques as we present them here are meant to be guide-lines for proofs by humans. However, they are chosen in such a way that they could be formally extended to form a complete formal system of predicate logic ("system of natural deduction") and even a complete system for automated theorem proving.

There are only very few different proof situations possible. Each of them is characterized by the syntactical structure of the sentence to be proven and by the syntactical structure of the sentences in the knowledge base. For choosing a particular proof technique one has to determine the "outermost construct" (quantifier, propositional connective, predicate or function symbol) of the sentence considered. For this, one has to have a firm knowledge of logical syntax in various disguise. For each of the constructs basically two different proof techniques are available depending on whether the sentence considered is in the knowledge base or whether it is the sentence to be proven.

Each proof technique describes how a given proof situation may be transformed to another "simpler" proof situation. The new proof situations are simpler because either the sentence to be proven has a simpler structure or more sentences are added to the knowledge base.

In "human" proofs, as presented in papers or talks, there are many ways of announcing the application of a certain proof technique. We will train the appropriate use of these idioms in the course. In the following brief summary of the proof techniques we only mention the most typical idioms for some of the proof techniques.

In the sequel,  $A, B, C$  are formulae,  $s$  and  $t$  are terms,  $P$  is a predicate constant,  $f$  is a function constant, and  $x$  is a variable.  $A[x]$  stands for a

formula  $A$  in which  $x$  occurs as free variable. Similarly,  $A[C]$  is a formula in which  $C$  occurs as subformula.

We assume that all free variables occurring in the sentences considered are universally bound before we apply any of the proof techniques. In particular, in all proof techniques for the “propositional connectives” (“not”, “and”, “or”, “implies”, “if and only if”) we assume that the formulae involved do not contain any free variables.

### The Four Basic Approaches

If we are supposed to prove a sentence  $A$  we actually do not know whether  $A$  is really true. We suggest to proceed as follows:

- Try to prove  $A$ . If you are successful be happy. Otherwise:
- Assume “not  $A$ ” and try to derive a contradiction. If you are successful be happy (you have proven  $A$ ). Otherwise:
- Try to prove “not  $A$ ”. If you are successful be happy (you have shown that one never should trust the boss). Otherwise:
- Assume  $A$  and try to derive a contradiction. If you are successful be happy (you have proven “not  $A$ ”). Otherwise:
- Start again with the attempt to prove  $A$ . (Don’t worry! You have gained a lot of new insight when you reach this stage. The second run will be much more successful.)

### Prove “for all $x$ , $A[x]$ ”

For proving

for all  $x$ ,  $A[x]$

show

$A[\bar{x}]$

where  $\bar{x}$  is a constant that did not occur so far.

One way of announcing the use of this proof technique is: "Let  $\bar{x}$  be arbitrary but fixed. We show  $A[\bar{x}]$ ." Sometimes one just assumes tacitly that, in the sequel,  $x$  is a (new) constant and one shows  $A[x]$ .

Use "for all  $x$ ,  $A[x]$ "

If

for all  $x$ ,  $A[x]$

is known then one may conclude

$A[t]$

where  $t$  is an arbitrary term.

One way of formulating this is: "Since we know that, for all  $x$ ,  $A[x]$  we also know that, in particular,  $A[t]$ ."

Prove "there exists  $x$  such that  $A[x]$ "

For proving

there exists  $x$  such that  $A[x]$

try to find a term  $t$  for which

$A[t]$

can be shown.

Finding a suitable term  $t$ , most times, is a non-trivial step in a proof, which needs creativity.

Use "there exists  $x$  such that  $A[x]$ "

If it is known that

there exists  $x$  such that  $A[x]$

and one has to prove

$B$

assume

$A[\bar{x}]$

where  $\bar{x}$  is a constant that did not occur so far and prove

$B$ .

One way of announcing the use of this proof technique is: "Let  $\bar{x}$  be such that  $A[\bar{x}]$ . We have to prove  $B$ ". Sometimes one just assumes tacitly that, in the sequel,  $x$  is a (new) constant and one shows  $B$ .

Prove "A and B"

For proving

$A$  and  $B$

prove

$A$

and prove

$B$ .

Use "A and B"

If one knows that

$A$  and  $B$

one knows

$A$

and one knows

$B$ .

Prove "A or B"

For proving

*A or B*

assume

not *A*

and prove

*B*

(or assume

not *B*

and prove

*A.*

Use "A or B"

If

*A or B*

is known and

*C*

should be proven assume

*A*

and prove

*C.*

Then assume

*B*

and prove

*C.*

One way of formulating the use of this technique is as follows: "We know *A or B* and want to prove *C*. *Case A*: We prove *C*. *Case B*: We prove *C*."



Prove "A implies B"

For proving

$A$  implies  $B$

assume

$A$

and prove

$B$ .

Use "A implies B"

As a general strategy, if

$B$

has to be proven, always look for sentences of the kind

$A$  implies  $B$

and prove

$A$ .

Prove "A if and only if B"

For proving

$A$  if and only if  $B$

assume

$A$

and prove

$B$ .

Then assume

$B$

and prove

$A$ .

Use "A if and only if B"

In a situation where

$$C[A]$$

has to be proven it may be helpful to look for sentences of the kind

$$A \text{ if and only if } B$$

and to try to prove

$$C[B].$$

This technique is often used in connection with "definitions" of predicate symbols, i.e. formulae of the form

$$P[x] \text{ if and only if } A[x]$$

Prove "not A"

For proving

$$\text{not } A$$

it is often helpful to assume

$$A$$

and to derive a contraction, i.e. to prove

$$\text{not } C$$

where

$$C$$

is in the knowledge base.

Prove " $P(t_1, \dots, t_n)$ "

For proving

$$P(t_1, \dots, t_n)$$

look for an "explicit definition" of the form

$$P(x_1, \dots, x_n) \text{ iff } A[x_1, \dots, x_n]$$

and prove

$$A[t_1, \dots, t_n].$$

Use " $P(t_1, \dots, t_n)$ "

If one knows that

$$P(t_1, \dots, t_n)$$

and an "explicit definition"

$$P(x_1, \dots, x_n) \text{ if and only if } A[x_1, \dots, x_n]$$

is in the knowledge base then

$$A[t_1, \dots, t_n]$$

may be added to the knowledge base.

Prove " $A[f(t_1, \dots, t_n)]$ "

For proving

$$A[f(t_1, \dots, t_n)]$$

where, for  $f$ , an "explicit definition" of the form

$$f(x_1, \dots, x_n) = s[x_1, \dots, x_n]$$

is available prove

$$A[s[t_1, \dots, t_n]].$$

For proving

$$A[f(t_1, \dots, t_n)]$$

where, for  $f$ , an “implicit definition” of the form

$$f(x_1, \dots, x_n) = \text{such a } y \text{ that } B[x_1, \dots, x_n, y]$$

is available prove

$$\text{for all } y, B[t_1, \dots, t_n, y] \text{ implies } A[y].$$

Use “ $A[f(t_1, \dots, t_n)]$ ”

If one knows

$$A[f(t_1, \dots, t_n)]$$

and an “explicit definition” of the form

$$f(x_1, \dots, x_n) = s[x_1, \dots, x_n]$$

is in the knowledge base then

$$A[s[t_1, \dots, t_n]]$$

may be added to the knowledge base.

If one knows

$$A[f(t_1, \dots, t_n)]$$

and an “implicit definition” of the form

$$f(x_1, \dots, x_n) = \text{such a } y \text{ that } B[x_1, \dots, x_n, y]$$

is in the knowledge base then the sentence

$$\text{there exists a } y \text{ such that } B[t_1, \dots, t_n, y] \text{ and } A[y]$$

may be added to the knowledge base.

### 3.8 Definitions Within Predicate Logic

Before using predicate logic as a working language one has to add one more facility which is indispensable for concise formalization, namely the facility of *defining new concepts* in terms of available concepts in predicate logic. In principle, definitions can be eliminated but practically they are indispensable because

- they allow to *make formulae shorter* and
- they are the means for *structuring knowledge* presented in predicate logic.

Practical mathematics would hardly be conceivable without definitions.

#### 3.8.1 The Four Basic Types of Definitions

There are four basic types of definitions available in predicate logic:

- (explicit) definitions of predicate symbols,
- explicit definitions of functions symbols,
- implicit non-unique definitions of function symbols,
- implicit unique definitions of function symbols.

We give one example for each of the four types.

In the examples we allow various simplifications of predicate logic notation! In particular we sometimes use natural language constructs for predicate and function symbols and allow phrases like “*i* is reducible” instead of “is-reducible(*i*)” etc. Also, we omit universal quantifiers at the beginning of a formula (i.e. free variables are considered to be universally quantified).

**Example 3.1** The following formula *defines the predicate symbol* “is reducible” (unary) using the binary predicate symbol  $|$  (“divides”):

$$i \text{ is reducible} : \leftrightarrow \forall f (f | i \rightarrow (f = 1 \vee f = i)).$$

The part of the definition to the left of the colon is called the “definiendum” (Latin: “to be defined”) and the right-hand part is called the “definiens” (Latin: “the defining”). The colon is *not* actually part of the definition. It is a symbol on the meta-level for indicating that this formula is a definition. In fact we will see that it is clear from the structure of the formula whether or not it is a definition.

**Example 3.2** The following formula *explicitly defines the 4-ary function symbol “determinant”* using the binary function symbols  $-$  and  $*$ :

$$\text{determinant}(a_{1,1}, a_{1,2}, a_{2,1}, a_{2,2}) := a_{1,1} * a_{2,2} - a_{1,2} * a_{2,1}.$$

**Example 3.3** The following formula *non-uniquely implicitly defines* (e.g. in the theory of complex numbers) the unary function symbol  $\sqrt{\phantom{x}}$  using the binary function symbols  $*$ :

$$\sqrt{x} := \text{a } y \text{ such that } y * y = x.$$

Here, the new quantifier “a ... such that ...” appears. Such a formula should just be conceived as an abbreviation for the following formula:

$$y = \sqrt{x} \rightarrow y * y = x.$$

**Example 3.4** The following formula *uniquely implicitly defines* (e.g. in the theory of real numbers) the unary function symbol  $\sqrt{\phantom{x}}$  using the binary function symbols  $*$  and the binary predicate symbol  $\geq$ .

$$\begin{aligned} \sqrt{x} := \text{the } y \text{ such that} \\ (x \geq 0 \rightarrow (y \geq 0 \wedge y * y = x)) \\ \wedge \\ (x < 0 \rightarrow y = 0). \end{aligned}$$

Here, the new quantifier “the ... such that ...” appears. Such a formula should be conceived as an abbreviation for the following formula:

$$\begin{aligned} y = \sqrt{x} \leftrightarrow \\ (x \geq 0 \rightarrow (y \geq 0 \wedge y * y = x)) \\ \wedge \\ (x < 0 \rightarrow y = 0). \quad \square \end{aligned}$$

When introducing new predicate and function symbols by definitions, two questions arise:

- Can the definitions introduce contradictions into a theory that so far was consistent?
- Can one derive essentially new theorems that so far were not derivable in the theory?

The answer to both questions is “no”. If certain rules for definitions are followed, definitions cannot introduce contradictions. Also, no “essentially” new theorems can be derived where “essentially new” means “new after elimination of the defined symbols”. In fact, defined symbols can be eliminated in the sense that each formula containing defined symbols can be systematically transformed into an equivalent formula not containing these symbols. The transformed formulae can already be derived in the old theory.

The proof of these facts about definitions is not really difficult but quite lengthy. We cannot give the proofs. However, we exactly formulate the logical facts about definitions so that practical situations can be handled in a clean way.

### 3.8.2 Theories

As we have seen in the above examples of implicit definitions, it is important to consider the underlying theory for deciding whether or not a definition is appropriate. Hence, for making the formal properties of definitions clear we need the notion of a “theory”, i.e. knowledge formulated in predicate logic. “Theories” are characterized by the language in which they are formulated, i.e. by a domain of symbols, and by their “knowledge base” or “set of axioms”, i.e. by a set of formulae from which all other facts of the theory can be derived by reasoning.

Let  $V$  be a fixed set of variables.

**Definition 3.5 (Theory)**  $T$  is called a theory of first order predicate logic iff there exist  $S$  and  $F$  such that  $T = (S, F)$  and

$S$  is a domain of non-logical constants disjoint from  $V$  and  
 $F$  is a set of formulae over  $V$  and  $S$ .

**Definition 3.6 (Theorem)**  $f$  is a theorem of a theory  $(S, F)$  iff  $F \vdash_{ND} f$ .

**Definition 3.7 (Extension)** The theory  $T' = ((FS', RS', AR'), F')$  is an extension of the theory  $T = ((FS, RS, AR), F)$  iff

$FS \subseteq FS'$ ,  
 $RS \subseteq RS'$ ,  
 for all  $fs \in FS$ ,  $AR(fs) = AR'(fs)$ ,  
 for all  $rs \in RS$ ,  $AR(rs) = AR'(rs)$ ,  
 for all  $f$ ,  
 if  $f$  is a theorem of  $T$  then  $f$  is a theorem of  $T'$ .  $\square$

The latter condition can also be replaced by the condition that all  $f$  in  $F$  must be theorems in  $T'$ . (However, it is not necessary that  $F \subseteq F'$ .)

**Definition 3.8 (Conservative Extension)** The theory  $T' = (S', F')$  is a conservative extension of the theory  $T = (S, F)$  iff  $T'$  is an extension of  $T$  and

for all formulae  $f$  over  $S$ ,  
 if  $f$  is a theorem of  $T'$  then  $f$  is a theorem of  $T$ .  $\square$

Let, for the next four subsections  $T = (S, F)$ , with  $S = (FS, RS, AR)$ , be an arbitrary but fixed theory.

### 3.8.3 Definitions of Predicate Symbols

**Definition 3.9 (Form of Explicit Definitions)** A formula  $d$  is a definition of the predicate symbol  $rs$  over the theory  $T$  iff  $d$  has the form

$$\forall v_1, \dots, v_n (rs(v_1, \dots, v_n) \leftrightarrow f)$$

where  $rs$  is an  $n$ -ary relation symbol not occurring in  $RS$  and  $f$  is a formula over  $S$  in which no variables other than the distinct variables  $v_1, \dots, v_n$  are free.

**Definition 3.10 (Translated Formula)** Let  $T' := (S', F')$  be the extension of  $T$ , where

$$\begin{aligned}
 S' &:= (FS, RS \cup \{rs\}, AR'), \\
 F' &:= F \cup \{d\}, \text{ and} \\
 d &\text{ is a definition of } rs \text{ over } T \text{ of the form} \\
 &rs(v_1, \dots, v_n) \leftrightarrow f.
 \end{aligned}$$

Then, for each formula  $g'$  over  $S'$ , the translation of  $g'$  is the formula  $g$  that results from  $g'$  by replacing each part  $rs(t_1, \dots, t_n)$  by  $f[(v_1, \dots, v_n) \leftarrow (t_1, \dots, t_n)]$ .



**Lemma 3.11 (Elimination of Defined Symbol)** With the notation of the previous definition we have:

$$\begin{aligned}
 F' \vdash_{ND} (g' \leftrightarrow g), & \tag{1} \\
 T' \text{ is a conservative extension of } F, & \tag{2} \\
 F' \vdash_{ND} g' \text{ iff } F \vdash_{ND} g. & \tag{3}
 \end{aligned}$$

□

(3) means that the defined predicate symbol  $\tau s$  may always be “eliminated” from any formula  $g'$  such that the resulting formula is derivable in  $F$  iff  $g'$  is derivable in  $F'$ . (3) is an easy consequence of (1) and (2).

Note that the condition on the variables is crucial in definitions of predicate symbols.

**Example 3.12** The “definition”

$$f \text{ is a factor} :\leftrightarrow f \mid x$$

introduces a contradiction (i.e. the extension of the given theory by this definition is not “conservative”), namely

$$\begin{aligned}
 3 \text{ is a factor} & :\leftrightarrow 3 \mid 3, \text{ and} \\
 3 \text{ is a factor} & :\leftrightarrow 3 \mid 5.
 \end{aligned}$$

Hence, 3 is a factor because  $3 \mid 3$ , and 3 is not a factor, because not  $3 \mid 5$ , a contradiction. □

In practice, the definitions of predicate symbols are sometimes also given in a form where the left-hand side does not only contain variables but terms. This must be handled with care and is only possible if the occurring terms have the properties of injective “pairing functions”. In fact we used this type of definition several times on the “metalevel”.

**Example 3.13**

$$\begin{aligned}
 (S, F) \text{ is a theory} & :\leftrightarrow \\
 S \text{ is a domain of symbols} \wedge F \text{ is a set of formulae.} &
 \end{aligned}$$

“Is a theory” is a unary predicate symbol (on the metalevel), “ $(S, F)$ ” is a term, not a variable!. Such a “definition” cannot introduce any contradiction because for the function symbol “ $(,)$ ” in set theory the following property holds:

$$(x, y) = (x', y') \leftrightarrow x = x' \wedge y = y'. \quad (\text{uniqueness})$$

Hence  $S$  and  $F$  are uniquely determined by  $(S, F)$ . Therefore it is not possible to derive the contradiction that something is a theory and is not a theory. In fact the above "definition" can always be rewritten in the following form (which, however, is clumsy):

$$\begin{aligned} T \text{ is a theory } &:\leftrightarrow \\ &\exists S, F (T = (S, F) \wedge S \text{ is a domain of symbols } \wedge \\ &F \text{ is a set of formulae}). \quad \square \end{aligned}$$

A "definition" having terms on the left-hand side is not allowed if the uniqueness property cannot be proven in the theory. Consider the following example:

**Example 3.14**

$$x + y \text{ is a nice sum } \leftrightarrow x = 2 * y.$$

In fact, this definition leads to a contradiction:

$$\begin{aligned} 6 + 3 \text{ is a nice sum } &\leftrightarrow 6 = 2 * 3, \\ 5 + 4 \text{ is a nice sum } &\leftrightarrow 5 = 2 * 4. \end{aligned}$$

Hence, 9 is both a nice sum and not a nice sum.

### 3.8.4 Properties of Explicit Definitions of Function Symbols

**Definition 3.15 (Form of Explicit Definitions)** A formula  $d$  is a definition of the function symbol  $fs$  over the theory  $T$  iff  $d$  has the form

$$\forall v_1, \dots, v_n ( fs(v_1, \dots, v_n) := t )$$

where  $fs$  is an  $n$ -ary function symbol not occurring in  $FS$  and  $t$  is a term over  $S$  in which no variables other than the distinct variables  $v_1, \dots, v_n$  are free.  $\square$

The notion of the translated formula  $g$  of a formula  $g'$  involving the new function symbol  $fs$  and the way the new function symbol can be eliminated is exactly analogous to the case of defined predicate symbols.

Again, the condition on the variables is crucial in the explicit definitions of function symbols.

**Example 3.16** The “definition”

$$f(x) := x * y$$

introduces a contradiction. Why?  $\square$

Again, the “explicit” definitions of function symbols are sometimes also given in a form where the left-hand side does not only contain variables but terms. This is again possible if the occurring terms have the properties of injective “pairing functions”.

**Example 3.17**

On the metalevel we could define

$$\begin{aligned} \langle (t_1 \equiv t_2) \rangle_A &:= \text{equ}(\langle t_1 \rangle_A, \langle t_2 \rangle_A), \\ x = y \rightarrow \text{equ}(x, y) &:= \text{T}, \\ \neg x = y \rightarrow \text{equ}(x, y) &:= \text{F}. \end{aligned}$$

Here, we used  $\equiv$  instead of  $=$  for denoting equality on the object level in order to distinguish it from the  $=$  on the metalevel. Strictly, this is not an explicit definition. However, it can easily be transformed into the following explicit definition of the binary function symbol “ $\langle, \rangle$ ” because the components  $t_1$  and  $t_2$  are uniquely determined in the formula  $t_1 \equiv t_2$ . (“ $\equiv$ ” is a function symbol on the metalevel!).

$$\langle f \rangle_A := \text{equ}(\langle \text{op}_1(f) \rangle_A, \langle \text{op}_2(f) \rangle_A).$$

Here,  $\equiv$  again has the essential “pairing function property”

$$(x \equiv y) = (x' \equiv y') \leftrightarrow (x = x') \wedge (y = y')$$

and  $\text{op}_1, \text{op}_2$  are the corresponding “projection functions” satisfying

$$\begin{aligned} \text{op}_1(x \equiv y) &= x, \\ \text{op}_2(x \equiv y) &= y, \\ \text{op}_1(z) \equiv \text{op}_2(z) &= z. \quad \square \end{aligned}$$

### 3.8.5 Properties of Non-Unique Implicit Definitions of Function Symbols

**Definition 3.18 (Form of Definitions)** A formula  $d$  is a non-unique implicit definition of the function symbol  $fs$  over the theory  $T$  iff  $d$  has the form

$$\forall v_1, \dots, v_n, v (v = fs(v_1, \dots, v_n) \rightarrow f)$$

where  $fs$  is an  $n$ -ary function symbol not occurring in  $FS$  and  $f$  is a formula in which no variables other than the distinct variables  $v_1, \dots, v_n, v$  are free.  $\square$

Note that  $d$  is equivalent to

$$f[v \leftarrow fs(v_1, \dots, v_n)].$$

We should also mention that a formula  $d$  of the above kind is not always considered a "definition" of the function symbol. Some authors prefer to reserve the word "definition" for what we call here "unique implicit definitions" (see next subsection). In fact it is not possible to eliminate non-uniquely defined function symbols. Still, the expressive power of a theory is not essentially enhanced by non-uniquely defined function symbols:

**Lemma 3.19 (Extension is Conservative)** Let  $T' := (S', F')$  be the extension of  $T$ , where

$$S' := (FS, RS \cup \{rs\}, AR'),$$

$$F' := F \cup \{d\}, \text{ and}$$

$d$  is a non-unique implicit definition of  $fs$  over  $T$  of the form

$$\forall v_1, \dots, v_n, v (v = fs(v_1, \dots, v_n) \rightarrow f).$$

*if the formula  $\forall v_1, \dots, v_n \exists v (f)$  are Theo: (existence condition)*

Then  $F'$  is a conservative extension of  $F$ .  $\square$

It is crucial that the (existence condition) is a theorem in  $T$ . Otherwise, a contradiction could be introduced by an implicit definition.

**Example 3.20** Let

$$y = \sqrt{x} \rightarrow y * y = x$$

be a formula in which the variables range over the real numbers. Then

$$\sqrt{-1} * \sqrt{-1} = -1$$

and, hence,

$$\exists y(y * y = -1)$$

in contradiction to

$$\neg \exists y(y * y = -1),$$

which can be proven in the theory of real numbers.  $\square$

### 3.8.6 Properties of Unique Implicit Definitions of Function Symbols

**Definition 3.21 (Form of Definitions)** A formula  $d$  is a unique implicit definition of the function symbol  $fs$  over the theory  $T$  iff  $d$  has the form

$$\forall v_1, \dots, v_n, v(v = fs(v_1, \dots, v_n) \leftrightarrow f)$$

where  $fs$  is an  $n$ -ary function symbol not occurring in  $FS$  and  $f$  is a formula in which no variables other than the distinct variables  $v_1, \dots, v_n, v$  are free.

**Definition 3.22 (Translated Formula)** Let  $T' := (S', F')$  be the extension of  $T$ , where

$$S' := (FS, RS \cup \{fs\}, AR'),$$

$$F' := F \cup \{d\}, \text{ and}$$

$d$  is a unique implicit definition of  $fs$  over  $T$  of the form

$$\forall v_1, \dots, v_n, v(v = fs(v_1, \dots, v_n) \leftrightarrow f).$$

Then, for each atomic formula  $g'$  over  $S'$ , the translation of  $g'$  is the formula  $g$  that results from  $g'$  by recursive application of the following step: If  $fs$  does not occur in  $g'$  then  $g := g'$  otherwise  $g'$  can be written in the form

$$h[w \leftarrow fs(t_1, \dots, t_n)],$$

where  $h$  is an atomic formula,  $w$  is a variable and  $fs$  does not occur in any of the  $t_1, \dots, t_n$ . Then  $g$  is

$$\exists w(f[(v_1, \dots, v_n, v) \leftarrow (t_1, \dots, t_n, w)] \wedge h^*),$$

where  $h^*$  is the translation of  $h$ . (Note that  $h$  contains one less occurrence of  $fs$  than  $g'$ .)

The translation of non-atomic formulae proceeds by translating each atomic part.

**Lemma 3.23 (Elimination of Defined Symbol)** With the notation of the previous definition we have: If the formulae

$$\begin{aligned} \forall v_1, \dots, v_n \exists v(f), \text{ and} & \quad (\text{existence condition}) \\ \forall v_1, \dots, v_n, v, v'(f \wedge f[v \leftarrow v'] \rightarrow v = v'), & \quad (\text{uniqueness condition}) \end{aligned}$$

are theorems of  $T$  then

$$F' \vdash_{ND} (g' \leftrightarrow g), \tag{1}$$

$$F' \text{ is a conservative extension of } F, \tag{2}$$

$$F' \vdash_{ND} g' \text{ iff } F \vdash_{ND} g. \tag{3}$$

□

It is crucial that both the existence condition and the uniqueness condition is a theorem in  $T$ . Otherwise, a contradiction could be introduced by an implicit definition.

**Example 3.24** Let

$$\begin{aligned} y = \sqrt{x} & :\leftrightarrow \\ (x \geq 0 \rightarrow y * y = x) & \\ \wedge & \\ (x < 0 \rightarrow y = 0) & \end{aligned}$$

be a formula in which the variables range over the real numbers. The uniqueness condition is not satisfied. We obtain the following contradiction:

$$\begin{aligned} 3 = \sqrt{9} & \leftrightarrow \\ (9 \geq 0 \rightarrow 3 * 3 = 9) & \\ \wedge & \\ (9 < 0 \rightarrow 3 = 0), & \tag{1} \end{aligned}$$

$$\begin{aligned} -3 = \sqrt{9} & \leftrightarrow \\ (9 \geq 0 \rightarrow (-3) * (-3) = 9) & \\ \wedge & \\ (9 < 0 \rightarrow -3 = 0), & \tag{2} \end{aligned}$$

$$3 = \sqrt{9}, \tag{from (1)}$$

$$-3 = \sqrt{9}, \tag{from (2)}$$

$$3 = -3. \quad \square$$

Note that for definitions by “cases” in which each case has the form of an explicit definition, both the (existence condition) and the (uniqueness condition) are automatically satisfied.

**Example 3.25**

$$\begin{aligned} x = y &\rightarrow \text{equ}(x, y) = \mathbf{T}, \\ \neg(x = y) &\rightarrow \text{equ}(x, y) = \mathbf{F}, \end{aligned}$$

is such a definition. Usually, these definitions are written in the form

$$\text{equ}(x, y) = \begin{cases} \mathbf{T}, & \text{if } x = y, \\ \mathbf{F}, & \text{otherwise.} \quad \square \end{cases}$$

Finally, we give an example of the above translation process.

**Example 3.26 (Translation of a Formula)** Let  $d$  be the following unique implicit definition of the unary function symbol “reverse” (where the variable  $f$  ranges over sequences, for simplicity of fixed length 10, and  $i$  and  $j$  range over integers):

$$f' = \text{reverse}(f) :\leftrightarrow \forall i(1 \leq i \leq 10 \rightarrow f_i = f'_{11-i}).$$

Let  $g$  be the following formula:

$$\text{reverse}(\text{reverse}(f)) = f.$$

First step of translation:

$$\begin{aligned} \exists f' (f' = \text{reverse}(f) \wedge \text{reverse}(f') = f), \\ \exists f' (\forall i(1 \leq i \leq 10 \rightarrow f_i = f'_{11-i}) \\ \wedge \text{reverse}(f') = f). \end{aligned}$$

Second step of translation:

$$\begin{aligned} \exists f' (\forall i(1 \leq i \leq 10 \rightarrow f_i = f'_{11-i}) \\ \wedge \exists f'' (f'' = \text{reverse}(f') \wedge f'' = f)); \\ \exists f' (\forall i(1 \leq i \leq 10 \rightarrow f_i = f'_{11-i}) \\ \wedge \exists f'' (\forall i(1 \leq i \leq 10 \rightarrow f'_i = f''_{11-i}) \\ \wedge f'' = f)). \end{aligned}$$

## Chapter 4

# The Completeness Theorem

### 4.1 Reduction to Model Construction

We have seen that the set of correct inference rules used in mathematical practice can be reduced to very few rules, in fact, those compiled in the Natural Deduction Calculus of the previous chapter. Can there exist formulae  $f$  that are logically valid (i.e. formulae  $f$  that are true independent of the meaning of the symbols occurring in  $f$ ) but cannot be derived in this calculus? The surprising answer is: no. The proof of this result is quite involved and was first given by (Goedel 1930). The first step consists in a reduction of the question to the following theorem.

#### Theorem 4.1 (Model Existence)

If  $F$  is consistent then  $F$  is satisfiable.

*Proof:* The proof will be given in the subsequent sections. For proving this theorem one constructs a valuation (i.e. a domain, an interpretation, and an assignment) such that all formulae of  $F$  are true in this valuation. The main idea is that, as the "material" for constructing this domain, one takes the "syntactic material" available, namely the set of terms built from the symbols occurring in  $F$ .  $\square$

Having proven the above theorem it is then easy to show the following completeness theorem.



**Theorem 4.2 (Gödel's Completeness Theorem)**

→ holds for predicate logic  
of first order, not for  
higher-order predicate logic.

If  $F \models f$  then  $F \vdash_{ND} f$ .

*Proof:* Assume  $F \not\vdash_{ND} f$ . Then  $F \cup \{\neg f\}$  is consistent by Lemma 3.17. Hence, by the above Theorem on Model Existence,  $F \cup \{\neg f\}$  is satisfiable. Hence,  $f$  cannot be a logical consequence of  $F$ .  $\square$

Summarizing the Correctness Theorem and the Completeness Theorem, and the Consistency Theorem and the Model Existence Theorem, we obtain the following perfect parallelism between "semantical and syntactical reasoning" in first order logic.

**Theorem 4.3 (Semantical and Syntactical Reasoning)**

$F \models f$  iff  $F \vdash_{ND} f$ .

$F$  is satisfiable iff  $F$  is consistent.  $\square$

In terms of computer science, this fundamental result can be paraphrased by saying that, in the area of first order predicate logic, logical consequences can be established by finite sequences of correct inference steps each of which is so simple that

- it can be verified by a computer and
- it can be executed by considering only the syntactical structure of formulae without any appeal to the meaning of the symbols occurring.

## Chapter 5

# Predicate Logic as a Working Language

In this chapter we show some examples of how various branches of mathematics and computer science can be formalized within predicate logic.

In contrast to the preceding chapters in which we spoke *about* predicate logic, in the main sections of this chapter we will speak *in* predicate logic.

In the first two sections we give two “universal” examples of using predicate logic: set theory as a universal frame for mathematics and logic programming as a universal frame for computer science.

### 5.1 Set Theory as a First Order Predicate Logic Theory

#### 5.1.1 Underlying Intuition

For a given domain  $D$  and interpretation  $I$ , a formula  $f$  containing a free variable  $v$  is true or false depending on the value (object in  $D$ ) assigned by an assignment  $A$  to  $v$ . The formula is true for some objects in the domain and is false for others, i.e. the formula describes a property of objects in the domain.

**Example 5.1** With  $D := \mathbb{N}$ ,  $I(2) := \text{two}$ ,  $I(\cdot) := \text{multiplication}$ , and  $A$  arbitrary, the formula

$$\exists y x = 2 \cdot y$$

is true or false depending on the value assigned to  $x$ :

$$\begin{aligned} \langle \exists y x = 2 \cdot y \rangle Val_{\{x:=1\}} &= \mathbf{F}, \\ \langle \exists y x = 2 \cdot y \rangle Val_{\{x:=2\}} &= \mathbf{T}, \\ \langle \exists y x = 2 \cdot y \rangle Val_{\{x:=3\}} &= \mathbf{F}, \\ \langle \exists y x = 2 \cdot y \rangle Val_{\{x:=17\}} &= \mathbf{F}, \\ \langle \exists y x = 2 \cdot y \rangle Val_{\{x:=18\}} &= \mathbf{T}, \\ \dots & \end{aligned}$$

□

For constructing the various domains of mathematics, for example the domain of real numbers, it is often necessary to speak *about* such properties. For example, in the construction of real numbers one may want to say: "Let  $f_1$  and  $f_2$  be two *properties* (i.e. formulae with a free variable  $v$ ) such that, if  $f_1$  is true for  $A[v := x_1]$  and  $f_2$  is true for  $A[v := x_2]$ , then  $x_1 < x_2$ ". However, there is no construct in first order predicate logic for formulating assertions *about formulae* in predicate logic.

The following trick helps to overcome this deficiency: A binary symbol  $\in$  is introduced, which denotes a relation that is required to have the following property:

$$\text{for (almost) arbitrary formula } f \text{ containing a free variable } v, \\ \exists S \forall v (v \in S \leftrightarrow f). \quad (\text{separation axiom for } f)$$

If one reads " $v \in S$ " by " $v$  is contained in  $S$ " (or, "the object  $v$  is an element of the set  $S$ "), then the above separation axiom for  $f$  says that

"There is a set (an object)  $S$  that contains exactly the objects satisfying  $f$ ."

Hence, for describing this fundamental property of  $\in$  one needs infinitely many axioms, one for each formula  $f$ . In fact, we have seen in Chapter 1 that the separation axiom must not be postulated for completely arbitrary formulae  $f$  because contradictions may arise (Russell antinomy).

Also, some additional axioms are necessary that add some more features to the notion of  $\in$  and guarantee the possibility to "construct" certain sets. Examples of such axioms are:

- an axiom that excludes the existence of infinite chains  $\dots x_3 \in x_2 \in x_1$  of objects ("regularity axiom"),

- an axiom that guarantees the existence of at least one infinite set ("infinity axiom"), which allows the construction of the natural numbers and all the other infinite domains of mathematics,
- an axiom ("extensionality axiom") that postulates that two "properties" (i.e. "sets") are equal if they hold for the same objects (i.e. contain the same objects as elements; have the same "extension"),
- axioms that allow to construct the power set (set of all subsets) of a set, the "range" of relations etc.

The latter axioms must be formulated with caution. As we have seen, if one is too careless in requiring the existence of certain sets, contradictions may arise. As a guiding line, one only should require the existence of sets whose elements "are already available before the new set is formed". For example, the following modified version of the separation axiom

$$\exists S \forall v (v \in S \leftrightarrow v \in S' \wedge f)$$

can be postulated for any  $f$  without any restriction "because" the objects  $v$  that are collected in  $S$  are already available in the given  $S'$ .

When set theory is developed along the above lines, another fundamental construction is taken care of by the same token: For a given domain  $D$  and interpretation  $I$ , a term  $t$  containing a free variable  $v$  yields an object in the domain that depends on the value (object in  $D$ ) assigned by an assignment  $A$  to  $v$ , i.e. the term describes a construction (process, procedure) for obtaining output objects from input objects. Again, there is no possibility in first order predicate logic to formulate assertions *about* terms. For example, it is not possible to express

$$t \text{ is bounded} \leftrightarrow \exists b \forall v | t | \leq b$$

in first order predicate logic.

However, if  $\in$  has already been introduced in the way described above, then it is possible to define (by an implicit definition) a binary function symbol  $\odot$  ("application") in the theory such that for arbitrary terms  $t$  with free variable  $v$  the following holds:

$$\exists F \forall v (F \odot v = t).$$

↑  
"table lookup"

In this context, the object  $F$  is called a “function”.

$\in$  and  $\odot$  are the two fundamental notions of set theory, which simulate the static aspect of mathematics (the validity of facts, properties, relations) and the dynamic aspect of mathematics (the application of procedures, processes, constructions). However, one essential feature of these two aspects that is contained in the intuitive notion of “property” and “procedure” is lost in this set theoretic simulation of the two notions: Everything that has to do with the “intension” (i.e. with the linguistic formulation) of properties and procedures is dropped in the set theoretic formalization of the notions “property” and “procedure”. By the extensionality axiom, only the “extension” (i.e. domain of validity) of properties and the “extension” (i.e. input/output behavior) of procedures is retained. Thus, “set” and “function” are only very coarse, albeit useful, formalizations of the notions “property” and “procedure”.

### 5.1.2 The Axioms of Zermelo-Fraenkel Set Theory

There are various possibilities for formalizing the fundamental features of the notion  $\in$  by first order predicate logic axioms. We present here one particular set of axioms due to Zermelo and Fraenkel. This set of axioms is quite common and, in fact, is the one on which the comprehensive collection of mathematical results in the Bourbaki presentation of mathematics is based. In its formal parts, our presentation follows (Shoenfield 1967).

We fix the alphabet  $V := \{x, y, z, \dots\}$  that contains the ordinary variables used in mathematical texts. “ $v$ ” with subscripts is reserved as a metavariables for variables. As a convention, we omit the outermost universal quantifiers in the individual axioms. The domain  $S$  of symbols in the Zermelo-Fraenkel set theory consists of only one binary predicate symbol  $\in$ , i.e.  $S := (\{\}, \{\in\}, \text{AR})$ , where  $\text{AR}(\in) := 2$ .

For reading the axioms, it is appropriate to imagine that one speaks about a “world” in which there exist nothing else than “objects”, i.e. all variables occurring in set theoretic formulae are ranging just over “objects”. Between some of these objects  $x$  and  $y$  the relation  $x \in y$  (“ $x$  is contained in  $y$ ”, “the object  $x$  is element of the set  $y$ ”) holds. In this context the object  $y$  is called a “set”. However, note that the property “set” is *not* a concept of Zermelo-Fraenkel set theory, i.e. there is no relation symbol “is set” in Zermelo-Fraenkel set theory.

Axiom 5.2 (Extensionality)

$$\forall z(z \in x \leftrightarrow z \in y) \rightarrow x = y. \quad \square$$

The extensionality axiom states that two objects ("sets") are identical if they contain the same elements.

**Axioms 5.3 (Regularity)**

*an infinite bunch of axioms,  
so-called scheme of axioms  
(Axiomenschema)*

$$\exists y(y \in x) \rightarrow \exists y(y \in x \wedge \neg \exists z(z \in x \wedge z \in y)). \quad \square$$

The regularity axiom states that, if a set  $x$  contains an element, then it contains an element  $y$  that does not contain any element of  $x$ . (This axiom entails that there are no "infinite descending chains w.r.t.  $\in$ ").

**Axioms 5.4 (Subsets)** *also "comprehension"* All formulae of the following form are axioms:

$$\forall x \forall y \exists v_1 \forall v_2 (v_2 \in v_1 \leftrightarrow v_2 \in x \wedge f),$$

*note variable* *set* *This avoids Russell's Antinomy. (cf. page 12.)*

where  $f$  is a formula,  $v_1, v_2$ , and  $v_3$  are distinct variables, and  $v_1, v_2$  do not occur free in  $f$ .  $\square$

53

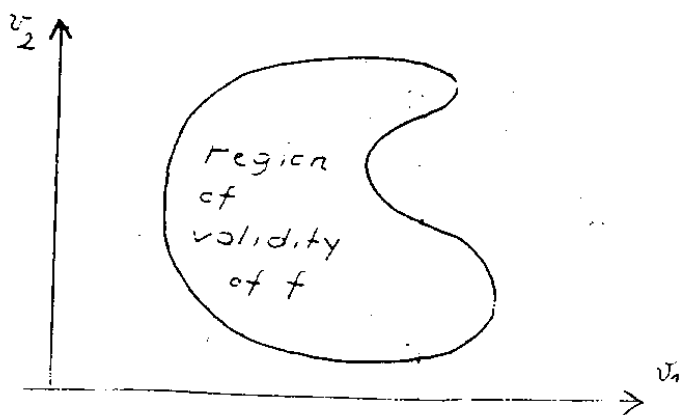
For a fixed  $f$ , the corresponding subset axiom states that for any set  $v_3$  there exists a set  $v_1$  that contains exactly those elements of  $v_3$  that have the property  $f$ .

**Axioms 5.5 (Replacement)** All formulae of the following form are axioms:

$$\forall v_1 \exists v_3 \forall v_2 (v_2 \in v_3 \leftrightarrow f) \rightarrow \exists v_4 \forall v_2 (\exists v_1 (v_1 \in v_5 \wedge f) \rightarrow v_2 \in v_4),$$

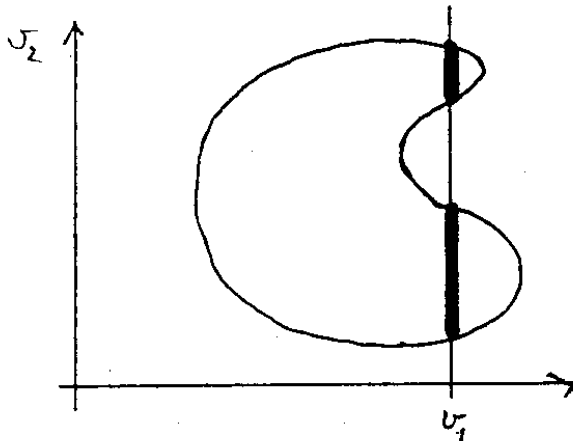
where  $f$  is a formula,  $v_1, v_2, v_3, v_4, v_5$  are distinct and  $v_3, v_4, v_5$  do not occur free in  $f$ .  $\square$

The meaning of these replacement axioms can be illustrated in a drawing. Typically, the formula  $f$  contains the free variables  $v_1$  and  $v_2$ . For certain pairs of values of  $v_1$  and  $v_2$  the formula  $f$  is true, for others it is false:

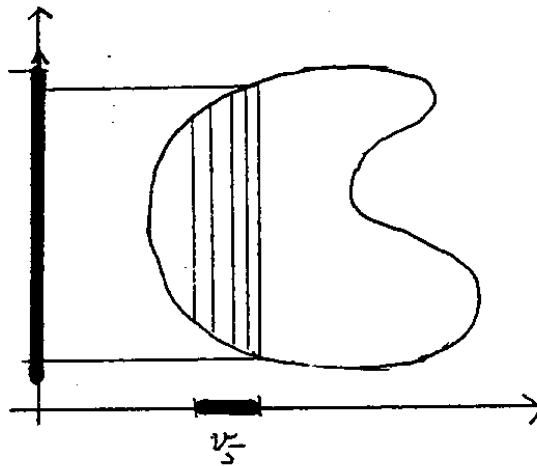


CHAPTER 5. PREDICATE LOGIC AS A WORKING LANGUAGE 11

The premise in the corresponding replacement axiom states that, for any value of  $v_1$ , the following set drawn as a bold vertical line exists:



The conclusion in the axiom then states that, for any given set  $v_5$  a set  $v_4$  of the kind shown as bold part of the  $v_2$ -axis exists (i.e. a set  $v_4$  containing all  $v_2$  that are in the relation  $f$  with a  $v_1$  in  $v_5$ ):



**Axioms 5.6 (Power Set)**

$$\exists w \forall y (\forall z (z \in y \rightarrow z \in x) \rightarrow y \in w). \quad \square$$

The power set axiom states that, for a given set  $x$ , there exists a set  $w$  that is big enough for containing all "subsets"  $y$  of  $x$ .

Axioms 5.7 (Infinity)

$$\exists x(\exists y(y \in x \wedge \forall z(z \notin y)) \wedge \forall y(y \in x \rightarrow \exists z(z \in x \wedge \forall w(w \in z \leftrightarrow (w \in y \vee w = y)))))) \quad \square$$

The infinity axiom states that there exists a set  $x$  that

- contains the “empty set” (i.e. a set that does not contain any element) and
- with each element  $y$  also contains an element  $z$  (the “successor of  $y$ ”) containing exactly all elements of  $y$  plus  $y$  as an element.

One can show that the empty set and its successors are all distinct, i.e. the axiom guarantees the existence of infinitely many objects.

### 5.1.3 The Set Braces

We want to show now

- that the well-known elementary concepts of set theory, like “ordered pair”, “function”, “natural number”, can be *defined* in Zermelo-Fraenkel set theory, and
- that the existence of the sets used in elementary set theory can be *proven* in this theory.

Before we do this, we show how the “set braces” can be introduced as abbreviating notation in Zermelo-Fraenkel set theory. For this we consider implicit definitions (of function symbols) of the following kind:

$$v_0 = fs(v_1) \leftrightarrow \forall v'_0(v'_0 \in v_0 \leftrightarrow f),$$

where  $f$  is a formula,  $v_0, v'_0$ , and  $v_1$  are distinct variables and the free variables of  $f$  are among  $v_1$  and  $v'_0$ . (All what follows can also be applied to  $n$ -ary function symbols  $fs$  for arbitrary  $n$ .)

Such a definition will be abbreviated by

$$fs(v_1) := \{v'_0 \mid f\},$$

(read: “ $fs(v_1)$  is the set of all  $v'_0$  for which  $f$ ”).

The uniqueness condition for such an implicit definition reads as follows:



$$\forall v'_0(v'_0 \in v_{01} \leftrightarrow f) \wedge \forall v'_0(v'_0 \in v_{02} \leftrightarrow f) \rightarrow v_{01} = v_{02}.$$

This condition is always true. Namely, from the assumption we can conclude that, for arbitrary  $v'_0$ ,

$$v'_0 \in v_{01} \text{ iff } f \text{ iff } v'_0 \in v_{02}$$

and, hence, by the extensionality axiom

$$v_{01} = v_{02}.$$

The existence condition for the above implicit definition is:

$$\forall v_1 \exists v_0 \forall v'_0(v'_0 \in v_0 \leftrightarrow f). \quad (\text{existence condition})$$

This condition is already satisfied if the following, seemingly weaker, condition is true:

$$\forall v_1 \exists v_0 \forall v'_0(v'_0 \in v_0 \leftarrow f). \quad (\text{weak existence condition})$$

In fact, if for an arbitrary  $v_1$ , by the weak existence condition,  $v_0$  is chosen such that

$$\forall v'_0(v'_0 \in v_0 \leftarrow f) \quad (*)$$

then one can apply the corresponding subset axiom

$$\exists v''_0 \forall v'_0(v'_0 \in v''_0 \leftrightarrow v'_0 \in v_0 \wedge f).$$

This axiom guarantees the existence of an  $v''_0$  such that, for all  $v'_0$ ,

$$v'_0 \in v''_0 \text{ iff } (v'_0 \in v_0 \wedge f) \underbrace{\text{iff}}_{\text{by } (*)} f.$$

The above weak existence condition is also called "set existence condition for  $f$ ".

One also allows the use of

$$\{v \mid f\}$$

as a part of formulae without mentioning the corresponding implicit definition of a function symbol. For example,

$$8 \in \{n \mid \exists m(n = 2 \cdot m)\}$$

has to be considered as an abbreviation for

$$S := \{n \mid \exists m(n = 2 \cdot m)\}, \\ 8 \in S.$$

The set braces have the characteristics of a quantifier: They take a formula and a variable and form a term out of it.  $\{v \mid f\}$  is a term that has the same free variables as  $f$  except for  $v$ , which becomes bound by the set braces.

Furthermore, one uses the following abbreviation:

$$\{t \mid f\}_v \text{ (or simply } \{t \mid f\} \text{ if } v \text{ is clear from the context)}$$

is an abbreviation for

$$\{v' \mid \exists v(v' = t \wedge f)\}, \text{ where } v' \text{ is a variable} \\ \text{that does not occur among the free variables of } t \text{ and } f.$$

For example,

$$\{2 \cdot n \mid a \leq n \leq b\}$$

is an abbreviation for

$$\{m \mid \exists n(m = 2 \cdot n \wedge a \leq n \leq b)\}.$$

Again, the uniqueness condition for definitions involving this type of set braces is always satisfied. For satisfying the existence condition

$$\exists v'' \forall v' (\exists v(v' = t \wedge f) \rightarrow v' \in v'')$$

(where  $v''$  is a "new" variable) it suffices that

$$\exists v''' \forall v(f \rightarrow v \in v''').$$

(where  $v'''$  is a "new" variable). Roughly, this means that, for showing that a set containing the "range" of a term  $t$  exists, it suffices to show that a set containing the "domain" of the term exists. We omit the proof, which essentially involves the replacement axiom.

### 5.1.4 First Stages of Developing Set Theory

First, one can introduce the concept of "subset" by the explicit definition

$$x \subseteq y \text{ (read: "x is subset of y")} := \forall z(z \in x \rightarrow z \in y).$$

Then, the notion of "power set" is introduced by the implicit definition

$$P(x) \text{ (read: "the power set of x")} := \{y \mid y \subseteq x\}.$$

The set existence condition for this definition is

$$\forall x \exists w \forall y (y \subseteq x \rightarrow y \in w),$$

which is exactly the power set axiom, i.e. the condition is trivially provable in the theory.

The empty set is defined by

$$\emptyset \text{ (read: "the empty set")} := \{x \mid x \neq x\}.$$

The set existence condition for this definition

$$\exists y \forall x (x \neq x \rightarrow x \in y)$$

follows from  $x = x$ . ( $x = x$  implies  $\forall y \forall x (x \neq x \rightarrow x \in y)$  from which the above set existence condition follows immediately.)

The unordered pair is defined by

$$\{x, y\} \text{ (read: "the unordered pair consisting of x and y")} := \{z \mid z = x \vee z = y\}.$$

Note that, here, the braces are binary function symbols and, formally, have nothing to do with the set braces used as a quantifier.

For proving the set existence condition for this implicit definition we proceed as follows. First we define a new function symbol  $F$ :

$$\begin{aligned} F(\emptyset, x, y) &:= x, \\ F(z, x, y) &:= y \leftarrow z \neq \emptyset. \end{aligned}$$

Now it can be shown that

$$\forall z (z = x \vee z = y \rightarrow z \in \{F(z) \mid z \in P(P(\emptyset))\}).$$

$\downarrow$   
 $x, y$

(The details of this proof are left as an exercise.)

Having unordered pairs one can define singletons:

$$\{x\} \text{ (read: "the singleton, or unit set, consisting of } x\text{")} := \{x, x\}.$$

Note that the braces obtain yet another meaning here. Namely, they serve also as a unary function symbol.

Next one defines

$$\bigcup(x) \text{ (read: "the union of } x\text{")} := \{y \mid \exists z(z \in x \wedge y \in z)\}.$$

The existence condition is

$$\forall x \exists w \forall y (\exists z (z \in x \wedge y \in z) \rightarrow y \in w).$$

This condition can be derived from

$$\forall z \exists v \forall y (y \in v \leftrightarrow y \in z)$$

by the appropriate replacement axiom.

The next notion is defined explicitly:

$$x \cup y \text{ (read: "the union of } x \text{ and } y\text{")} := \bigcup(\{x, y\}).$$

The next two notions are defined implicitly in the form  $\dots := \{z \mid z \in x \wedge \dots\}$ . The existence condition in such cases is always satisfied (because of the subset axioms).

$$\begin{aligned} x \cap y \text{ (read: "the intersection of } x \text{ and } y\text{")} &:= \{z \mid z \in x \wedge z \in y\}, \\ x - y \text{ (read: "the set difference of } x \text{ and } y\text{")} &:= \{z \mid z \in x \wedge z \notin y\}. \end{aligned}$$

The next concept can be defined explicitly:

$$\langle x, y \rangle \text{ (read: "the ordered pair consisting of } x \text{ and } y\text{")} := \{\{x\}, \{x, y\}\}.$$

Ordered pairs have the following basic property:

$$\langle x, y \rangle = \langle x', y' \rangle \leftrightarrow x = x' \wedge y = y' \quad (\text{uniqueness of pair components}).$$

The proof is left as an exercise. The uniqueness of pair components guarantees that the following implicit definitions of the "projections"  $\pi_1$  and  $\pi_2$  are sound:

$$\begin{aligned} \pi_1(z) &:= \text{the } x \text{ such that} \\ &\quad (\exists x, y (z = \langle x, y \rangle) \rightarrow \exists y (z = \langle x, y \rangle)) \\ &\quad \wedge \\ &\quad (\neg \exists x, y (z = \langle x, y \rangle) \rightarrow x = \emptyset), \\ \pi_2(z) &:= \text{the } y \text{ such that} \\ &\quad (\exists x, y (z = \langle x, y \rangle) \rightarrow \exists x (z = \langle x, y \rangle)) \\ &\quad \wedge \\ &\quad (\neg \exists x, y (z = \langle x, y \rangle) \rightarrow y = \emptyset). \end{aligned}$$

The case of the definition that handles objects  $z$  that are not pairs is quite arbitrary. However, for the sake of uniqueness we have to specify some value of the projections in this case.

Now, the notion of Cartesian product can be based on the notion of ordered pairs:

$$\begin{aligned} x \times y \text{ (read: "the Cartesian product of } x \text{ and } y\text{")} &:= \\ &\{z \mid \exists a, b (a \in x \wedge b \in y \wedge z = \langle a, b \rangle)\}. \end{aligned}$$

The existence condition for this definition is satisfied: If  $a \in x$  and  $b \in y$  then both  $\{a\}$  and  $\{a, b\}$  are in  $P(x \cup y)$ . Hence,  $\langle a, b \rangle \in P(P(x \cup y))$ , i.e. there exists a set that contains all elements  $z$  appearing in the above definition.

By induction on  $n$  (on the metalevel) one can now introduce infinitely many function symbols  $\langle, \rangle$  and  $\times$  (one for each arity  $n$ ) denoting  $n$ -tuples and  $n$ -ary Cartesian products:

$$\begin{aligned} \langle x_1, \dots, x_n \rangle &= \langle x_1, \langle x_1, \dots, x_n \rangle \rangle, \\ x_1 \times \dots \times x_n &:= x_1 \times (x_2 \times \dots \times x_n). \end{aligned}$$

The definition of the corresponding projection functions  $\pi_i^n$  is left as an exercise.

Now the concept of "function" can be formally defined within set theory. As explained earlier in this chapter, in set theory a function intuitively is a set of pairs  $\langle a, b \rangle$  with the idea that  $b$  is the "output" of the function when applied to the "input"  $a$ . Formally, one can proceed as follows:

$$\begin{aligned} \text{Dom}(x) \text{ (read: "the domain of } x\text{")} &:= \{\pi_1(y) \mid y \in x\}, \\ \text{Rg}(x) \text{ (read: "the range of } x\text{")} &:= \{\pi_2(y) \mid y \in x\}, \\ \text{Fun}(x) \text{ (read: " } x \text{ is a function")} &:= \leftrightarrow \\ &\quad x \subseteq \text{Dom}(x) \times \text{Rg}(x), \end{aligned}$$

$$\begin{aligned}
 & \forall a, b, b' (\langle a, b \rangle, \langle a, b' \rangle \in x \rightarrow b = b'), \\
 \text{InjFun}(x) \text{ (read: "x is an injective function")} & := \leftrightarrow \\
 & \text{Fun}(x) \wedge \\
 & \forall a, a', b (\langle a, b \rangle, \langle a', b \rangle \in x \rightarrow a = a'), \\
 x \odot a \text{ (read: "the result of applying x to a")} & := \text{the } b \text{ such that} \\
 & ((\text{Fun}(x) \wedge \exists b (\langle a, b \rangle \in x)) \rightarrow \langle a, b \rangle \in x), \\
 & \wedge \\
 & (\neg(\text{Fun}(x) \wedge \exists b (\langle a, b \rangle \in x)) \rightarrow b = \emptyset).
 \end{aligned}$$

Now all the notions related with functions can be introduced in the usual manner.

### 5.1.5 The Natural Numbers in Set Theory

↓ for set

The set construction mechanisms of the preceding subsection are a powerful tool for building up new domains (sets together with functions operating on these sets) from existing domains. However, thus far we have not yet shown that there exist interesting basic domains from the tower of domains used in mathematics could be built up.

The domain of natural numbers is the crucial non-trivial domain that must be built up first. Every other domain (for example, the real and complex numbers, the domain of real functions, the domain of linear mappings between two vector spaces etc) can then be constructed using the construction mechanisms of the preceding subsection.

Therefore, in this subsection, we show how the natural numbers can be constructed *within* set theory. This means that, within set theory, a set  $N_0$  and an element 0 and a function  $S$  is *defined* such that the fundamental properties of the domain of natural numbers can be *proven*. The most important fundamental properties are the so called "Peano properties" ("Peano axioms"):

$$\begin{aligned}
 & 0 \in N_0, \\
 & x \in N_0 \rightarrow S(x) \in N_0, \\
 & S(x) \neq 0, \\
 & S(x) = S(y) \rightarrow x = y, \\
 & (0 \in x \wedge \forall x \in N_0 (x \in N \rightarrow S(x) \in N)) \rightarrow N_0 \subseteq N.
 \end{aligned}$$

In set theory, the construction of the domain of natural numbers is carried out in a very general setting: First, the notion of an "ordinal" is in-

roduced. Ordinals can be compared. Among two ordinals it is always clear which one is "earlier" and which one is "later". Ordinals always have an immediate successor. However, not all ordinals have an immediate predecessor. Such ordinals are called "limit ordinals". The set of natural numbers is then defined to be the "first limit ordinal", i.e. the first ordinal after all the successors of the ordinal "zero".

$$\begin{aligned}\text{Trans}(x) \text{ (read: "x is transitive")} &: \leftrightarrow \forall y(y \in x \rightarrow y \subseteq x), \\ \text{Ord}(x) \text{ (read: "x is an ordinal")} &: \leftrightarrow \text{Trans}(x) \wedge \forall y(y \in x \rightarrow \text{Trans}(y)).\end{aligned}$$

From now on, let  $\rho, \sigma, \tau$  be variables that range over ordinals. This convention facilitates writing complicated formulae. It should be clear how formulae using these "typed variables" can be viewed as abbreviations of ordinary predicate logic formulae.

As an exercise prove the following property of ordinals:

$$\begin{aligned}\text{Ord}(\emptyset), \\ x \in \sigma \rightarrow \text{Ord}(x).\end{aligned}$$

One now can define an ordering on ordinals:

$$\sigma < \tau : \leftrightarrow \sigma \in \tau.$$

As an exercise prove the following properties of " $<$ ":

$$\begin{aligned}\rho < \sigma \wedge \sigma < \tau &\rightarrow \rho < \tau, \\ \neg(\sigma < \sigma), \\ \neg(\sigma < \tau \wedge \tau < \sigma).\end{aligned}$$

(Hint: For the first property, use the transitivity of  $\tau$ . For the second property, prove first that  $x \notin x$ . The third property is a consequence of the first two.)

Using the regularity axiom one now can show the following fundamental minimality properties (one minimality property for each formula  $f$ ):

$$\exists \sigma f \rightarrow \exists \sigma (f \wedge \forall \tau (\tau < \sigma \rightarrow \neg f[\sigma \leftarrow \tau])).$$

In fact, one can even show that the "minimal" ordinal in the above property is uniquely defined:

$$\begin{aligned} & (f \wedge \forall \tau (\tau < \sigma \rightarrow \neg f[\sigma \leftarrow \tau])) \\ & \wedge \\ & (f[\sigma \leftarrow \sigma'] \wedge \forall \tau (\tau < \sigma' \rightarrow \neg f[\sigma \leftarrow \tau])) \\ & \rightarrow \\ & \sigma = \sigma'. \end{aligned}$$

It is therefore possible to introduce the quantifier "the first ordinal ... such that ..." for abbreviating formulae involving the above construction. For example,

$$\rho := \text{the first } \tau \text{ such that } \sigma < \tau,$$

can be used as an implicit definition as soon as one has proven

$$\forall \sigma \exists \tau (\sigma < \tau)$$

and is an abbreviation for

$$\rho := \text{the } \tau \text{ such that } \sigma < \tau \wedge \forall \tau' < \tau (\sigma \not< \tau').$$

Using these minimality properties one can prove one more property of " $<$ ":

$$\sigma < \tau \vee \sigma = \tau \vee \tau < \sigma,$$

which shows that there are no "incomparable" ordinals. Furthermore, the following powerful induction principle (the "*principle of transfinite induction*") can be proven, which again holds for arbitrary formulae  $f$ :

$$\forall \sigma (\forall \tau (\tau < \sigma \rightarrow f[\sigma \leftarrow \tau]) \rightarrow f) \rightarrow \forall \sigma f.$$

This principle allows to conclude  $\forall \sigma f$  if  $f$  can be proven under the "induction hypothesis" that, for all  $\tau < \sigma$ ,  $f$  holds for  $\tau$ .

We now can proceed to some preliminaries that later can be specialized for constructing the natural numbers:

$$S(\sigma) \text{ (read: "the successor of } \sigma\text{")} := \sigma \cup \{\sigma\}.$$

$S$  has the properties:

$$\begin{aligned} & \text{Ord}(S(\sigma)), \\ & S(\sigma) \neq \tau \text{ = the first ordinal } \tau \text{ such that } \sigma < \tau, \\ & S(\sigma) = S(\tau) \rightarrow \sigma = \tau, \end{aligned}$$

$$\begin{aligned} 0 &= \emptyset \\ 1 &= \{\emptyset, \{\emptyset\}\} \\ 2 &= \{\emptyset, \{\emptyset, \{\emptyset, \{\emptyset\}\}\} \end{aligned}$$



The following limit construction allows to go beyond iterated successors:

$$\begin{aligned} \sigma \text{ is a limit ordinal} &:\leftrightarrow \sigma \neq \emptyset \wedge \neg \exists \tau (\sigma = S(\tau)), \\ \forall y (y \in x \rightarrow \text{Ord}(y)) &\rightarrow \text{Ord}(\bigcup x), \\ \forall x \exists \sigma (\forall \tau (\tau \in x \rightarrow \tau < \sigma), & \\ \exists \sigma (\sigma \text{ is a limit ordinal}). & \end{aligned}$$

The last statement can be proven by using the infinity axiom that guarantees the existence of a set  $x$  containing  $\emptyset$  and  $S(y)$  whenever  $y \in x$ . Let  $z$  be the set of ordinals in  $x$ . Then  $\sigma := \bigcup z$  has the desired property.

Now one has all the ingredients for defining the *set of natural numbers*:

$$\begin{aligned} \omega \text{ (read: "the set of natural numbers")} &:= \\ &\text{the first ordinal } \sigma \text{ such that } \sigma \text{ is a limit ordinal.} \end{aligned}$$

Furthermore, we define

$$0 := \emptyset.$$

Now, one can show that that  $\omega$  together with 0 and  $S$  satisfies the fundamental "Peano properties" of natural numbers:

$$\begin{aligned} 0 \in \omega, \\ \sigma \in \omega \rightarrow S(\sigma) \in \omega, \\ S(\sigma) \neq 0, \\ S(\sigma) = S(\tau) \rightarrow \sigma = \tau, \\ (0 \in x \wedge \forall \sigma \in \omega (\sigma \in x \rightarrow S(\sigma) \in x)) \rightarrow \omega \subseteq x. \end{aligned}$$

The last property is the well-known and fundamental induction property of the set of natural numbers. In set theory this property can be *proven* (using the transfinite induction principle). One could introduce arbitrarily many constants for natural numbers by defining  $1 := S(0)$ ,  $2 := S(1)$ , ...

(Hence, ~~Dedekind~~'s remark that "God has created the natural numbers, everything else was created by men" could be paraphrased by "God has created the empty set, everything else was created by men".)

### 5.1.6 Developing All of Mathematics in Set Theory

Having the set  $\mathbb{N}_0 := \omega$  of natural numbers and the powerful mechanisms of set theory for constructing new sets and functions from given sets and

functions, one now can built up all domains of mathematics in the usual way.

For example, the set  $Z$  of integers can be obtained by constructing first  $N_0 \times N_0$  and setting  $Z := \{[p]_{\sim} \mid p \in N_0 \times N_0\}$ , where  $[p]_{\sim}$  is the equivalence class of  $p$  with respect to the equivalence relation  $(m, n) \sim (m', n') :\leftrightarrow m + n' = m' + n$ .

Similarly, the set  $Q$  of rational numbers can be constructed from the set  $Z$  by first forming the Cartesian product  $Z \times Z$  and then "factoring modulo a suitable equivalence relation".

Again similarly, albeit more involved, the set  $R$  of real numbers can be constructed from  $Q$  by first constructing the set of all sequences  $f : N_0 \rightarrow Q$  that have the "Cauchy-property" and then "factoring modulo a suitable equivalence relation".

The set  $C$  of complex numbers can be identified with the set  $R \times R$ . The arithmetical operations on  $C$  can be defined explicitly on such pairs.

The set of polynomials over a domain (with certain operations) can be identified with the set of finite sequences of "coefficients" from the given domain. The operations on polynomials can then be defined suitably.

The set of matrices over a domain  $D$  can be defined as the set of mappings  $m : \{1, \dots, m\} \times \{1, \dots, n\} \rightarrow D$ .

Accordingly, all domains of mathematics, including very abstract domains that result from collecting (infinitely) many other domains in a suitable way, can be constructed and their properties can be proven within set theory.

## 5.2 Predicate Logic as a Programming Language

### 5.2.1 Underlying Intuition

When an interpretation  $I$  in a domain  $D$  and an assignment  $A$  for the variables is fixed, a formula  $f$  of predicate logic describes a fact that may be true or false. If we leave  $A$  open, a formula describes a property of elements in  $D$  or a relation between elements in  $D$ .

In particular, formulae of predicate logic may describe properties of "desired objects", properties of "objects that should be constructed" or, in other words, formulae may describe *problems*.

**Example 5.1** The problem of "greatest common divisor" can be described by the following formula of predicate logic:

Given:  $x, y$ .

Find:  $z$  such that

$$z \mid x \wedge z \mid y \wedge \forall z'((z' \mid x \wedge z' \mid y) \rightarrow z' \leq z). \quad \square$$

The formula describes the property the desired object  $z$  should have in relation to the given  $x$  and  $y$ . (In fact, the formula itself is quite "indifferent" with respect to which free variable should stand for a "given" and which one for a "desired" object. The distinction between "given" and "desired" comes from "outside" the formula. The above formula also describes the (uninteresting) problem of finding  $x$  and  $y$  such that the given  $z$  is the greatest common divisor of  $x$  and  $y$ .)

In describing facts, properties and, in particular, problems, predicate logic serves a "static" purpose. Seemingly, this is far from the "dynamic" purpose programming languages have. Such languages do not describe facts but formulate instructions (procedures, algorithms) how to obtain certain objects.

Programs are written for solving problems, i.e. the objects constructed by programs should meet certain specifications. The output of programs should have certain properties in relation to the input. A program should meet its specification.

This is where the static aspect of logic and the dynamic aspect of programming languages come together. Logic can be used for specifying problems that should be solved by programs described in a programming language.

Facts described in logic can be proven or disproven. The proof of a formula is something outside the formula. A proof is a process, a sequence of steps. Hence, although the typical meaning of a logic formula is "static" the proof of a formula is "dynamic".

It is near at hand to ask: *Can the dynamic aspect of proving be used for making a programming language out of logic?* The answer is: Yes! In fact, computing can be seen as a particularly simple form of proving. Hence, theoretically, a part of logic is sufficient as a programming language. Practically, general proving is not very suitable as a computing mechanism because general proving is "undirected". For making proving practical as a computing mechanism one must restrict the enormous "search space" of general proving by allowing only certain sequences of proof steps.

Using (a restricted version of) proving as a computing mechanism has the wonderful consequence that problems and their solutions can be expressed in the same language. In fact, the exact problem specification is the solution when used as the initial input for a proof.

Also, another fundamental insight evolves when viewing logic as a programming language: Proofs become shorter when more "knowledge" (i. e. proven formulae) on the concepts involved in the problem specification is established. This leads to the conclusion that *the purpose of mathematical knowledge (theorems) ultimately is the speed-up of problem solving.*

**Example 5.2** All variables in this example range over the natural numbers. Let us define

$$\text{gcd}(x, y) := \text{the } z \text{ such that } z \mid x \wedge z \mid y \wedge \forall z'((z' \mid x \wedge z' \mid y) \rightarrow z' \leq z).$$

Essentially, "gcd" (greatest common divisor) abbreviates the problem specification in the previous example.

The following proof of

$$\text{gcd}(6, 8) = 2$$

can be viewed as a "computation" of the "output" 2 for the "input" 6 and 8. (In this computation we already presuppose that we have proven earlier that

$$a \mid b \rightarrow a \leq b$$

and various other well-known properties of " $\mid$ " and " $<$ ".)

$$\begin{aligned}
& z \mid 6 \rightarrow z \leq 6, \\
& z \mid 6 \rightarrow z = 1 \vee \dots \vee z = 6, \\
& 1 \mid 6, \\
& 2 \mid 6, \\
& 3 \mid 6, \\
& \neg(4 \mid 6), \\
& \neg(5 \mid 6), \\
& 6 \mid 6, \\
& z \mid 6 \rightarrow z = 1 \vee z = 2 \vee z = 3 \vee z = 6, \\
& \dots, \\
& z \mid 8 \rightarrow z = 1 \vee z = 2 \vee z = 4 \vee z = 8, \\
& z \mid 6 \wedge z \mid 8 \rightarrow z = 1 \vee z = 2, \\
& \dots, \\
& \gcd(6, 8) = z \rightarrow z = 2, \\
& \gcd(6, 8) = 2.
\end{aligned}$$

Of course, this “computation” is only one of the infinitely many possible traces of proofs of  $\gcd(6, 8) = 2$  and it needs the ingenuity of a human to propose a fast computational path in this indeterministic proof mechanism.

Adding the following knowledge about the functions  $\gcd$  and  $\text{rest}$  (which can be proven once and for all in a finite sequence of proof steps)

$$\begin{aligned}
& (x > y \wedge y \neq 0) \rightarrow \gcd(x, y) = \gcd(y, \text{rest}(x, y)), \\
& x < y \rightarrow \gcd(x, y) = \gcd(y, x), \\
& \gcd(x, 0) = x,
\end{aligned}$$

the above proof of  $\gcd(6, 8) = 2$  can be made much shorter and, what is more important, the necessary sequence of proof steps that leads to the desired “output” can be described by a simple rule that relieves the “prover” of any creative decisions:

$$\begin{aligned}
& \gcd(6, 8) = \\
& = \gcd(8, 6) = \\
& = \gcd(6, 2) = \\
& = \gcd(2, 0) = \\
& 2.
\end{aligned}$$

Of course, we left out many intermediate steps, notably the “computation” (proof) of the rest in the division of intermediate numbers. However, it is

easy to imagine that the division, as a proof procedure, can also be made efficient by introducing (proving) the knowledge:

$$\begin{aligned}x \geq y &\rightarrow \text{rest}(x, y) = \text{rest}(x - y, y), \\x < y &\rightarrow \text{rest}(x, y) = x.\end{aligned}$$

□

An analysis of the type of knowledge that lead to an efficient "computation" in the form of proofs shows that such "algorithmically useful knowledge" should have the form of "rewrite rules", i. e. formulae that do not involve any quantifiers and describe the reduction of a problem to other, "simpler", problems or the "recursive" reduction to the same problem for "smaller" input values.

Summarizing the intuition obtained from the above example, the goals that have to be achieved in order to make predicate logic a suitable programming language are:

- identification of a class of predicate logic formulae that describe "algorithmically useful knowledge" and reveal the recursive nature of such knowledge (see the Horn clauses in the next subsection),
- if possible, design of a procedure that transforms arbitrary formulae into formulae of the restricted class (see next subsection),
- specification of a restricted form of proving that is appropriate for the restricted class of formulae and relieves the prover of making decisions about directions in proof paths and achieves proof goals efficiently, i.e. with as few steps as a "deterministic" program in a conventional programming language would need (see "resolution proving" in the next but one subsection),
- if possible, proof that the restricted proving procedure is still complete, i. e. always finds a proof if the desired result is provable at all (i. e. provable with one of the procedures whose completeness is already known).

Essentially, these goals have been achieved by the various research efforts in computational logic (automated theorem proving, logic programming etc.) over the last three decades. We only sketch the result, mainly by examples. Extra courses of the RISC-curriculum are devoted to the details automated theorem proving and logic programming.

### 5.2.2 Clausal Form of Formulae

In an earlier chapter we have seen that every formula  $f$  can be transformed into a formula  $f'$  in Skolem normal form such that  $f$  is satisfiable iff  $f'$  is satisfiable. The procedure that achieves this, essentially proceeds in two steps:

- transformation into prenex form, i.e. a form in which all quantifiers appear as outermost symbols, by successive application of well-known equivalences like " $\neg\forall v f$  equivalent to  $\exists v \neg f$ ",
- transformation of the prenex form into Skolem normal form, i.e. a form in which no existential quantifiers occurs and all universal quantifiers appear as outermost quantifiers, by Skolemization steps by which, for example,  $\forall v_1 \exists v f$  can be transformed into  $\forall v_1 f[v \leftarrow fs(v_1)]$  with a new function symbol  $fs$ .

Formulae in Skolem normal form can be further normalized into "clausal form":

#### Definition 5.3 (Literals and Clauses)

$l$  is a literal iff

$l$  is an atomic formula or  $l = \neg f$ , where  $f$  is an atomic formula.

$C$  is a clause iff  $C$  is a finite set of literals.

$CF$  is a formula in clausal form iff  $CF$  is a finite set of clauses.  $\square$

A formula  $CF = \{C_1, \dots, C_m\}$  in clausal form, where each clause  $C_i = \{l_{i,1}, \dots, l_{i,n_i}\}$  and the  $l_{i,j}$  are literals, is an abbreviation for the formula

$$(l_{1,1} \vee \dots \vee l_{1,n_1}) \wedge \dots \wedge (l_{m,1} \vee \dots \vee l_{m,n_m}),$$

i. e.  $CF$  is a conjunction of disjunctions of literals.

A formula in Skolem normal form can be transformed into an equivalent clausal form by first omitting the universal quantifiers (this is only cosmetics; free variables are now viewed as being universally quantified) and then successively applying the following equivalences ("conjunctive normal form algorithm"):

$$(f_1 \leftrightarrow f_2) \text{ is equivalent to } ((f_1 \rightarrow f_2) \wedge (f_2 \rightarrow f_1)),$$

$$(f_1 \rightarrow f_2) \text{ is equivalent to } (\neg f_1 \vee f_2),$$

- $\neg\neg f$  is equivalent to  $f$ ,
- $\neg(f_1 \wedge f_2)$  is equivalent to  $(\neg f_1 \vee \neg f_2)$ ,
- $\neg(f_1 \vee f_2)$  is equivalent to  $(\neg f_1 \wedge \neg f_2)$ ,
- $(f_1 \vee (f_2 \wedge f_3))$  is equivalent to  $((f_1 \vee f_2) \wedge (f_1 \vee f_3))$ ,
- $((f_2 \wedge f_3) \vee f_1)$  is equivalent to  $((f_2 \vee f_1) \wedge (f_3 \vee f_1))$ .

An arbitrary proof problem, i.e. the question whether  $f_1 \dots f_n \vdash f$  (or, equivalently,  $f_1 \dots f_n \models f$ ), can also be transformed into the form of a satisfiability problem. Namely,

$f_1, \dots, f_n \models f$  iff  $f_1 \wedge \dots \wedge f_n \wedge \neg f$  is not satisfiable.

Also, one should note that for a formula  $f$  with free variables  $v_1, \dots, v_n$   $f$  is satisfiable iff  $\exists v_1, \dots, v_n f$  is satisfiable.

The following proposition summarizes all this.

**Theorem 5.4** There exists an algorithm (essentially consisting of prenex normalization, Skolemization, and conjunctive form simplification) that takes arbitrary formulae  $f_1, \dots, f_n, f$  and produces a formula  $f'$  such that

$f'$  is insatisfiable iff  $f_1 \dots f_n \models f$  and  $f'$  is in clausal form.

**Example 5.5** In the above example the computational problem "gcd(6, 8) = ?" can be formulated as the following proving problem (where we write all equations in predicate form, i.e. we write "gcd(x, y, z)" instead of "gcd(x, y) = z"):

$$\begin{aligned} \forall x, y, z (\text{gcd}(x, y, z) \leftarrow & \\ & \exists u (\text{gcd}(y, u, z) \wedge \text{rest}(x, y, u)) \wedge x \geq y) \wedge & \text{(case } x \geq y) \\ \forall x, y, z (\text{gcd}(x, y, z) \leftarrow & \text{gcd}(y, x, z) \wedge x < y) \wedge & \text{(case } x < y) \\ \forall x \text{ gcd}(x, 0, x) & & \text{(case } y = 0) \\ \rightarrow \exists z \text{ gcd}(6, 8, z). & & \text{(existence of result)} \end{aligned}$$

The computational problem is formulated as an existence proof: We are supposed to prove the existence of the gcd of 6 and 8. A "good" (i.e. constructive) proof will prove the existence by giving an example of an answer. Hence it will "prove", i. e. compute, the result.

The above formula is true (provable) iff the following formula leads to a contradiction (is insatisfiable, inconsistent):



$$\begin{array}{ll}
 \dots \wedge & \text{(case } x \geq y) \\
 \dots \wedge & \text{(case } x < y) \\
 \dots \wedge & \text{(case } y = 0) \\
 \neg \exists z \text{gcd}(6, 8, z). &
 \end{array}$$

The clausal form of the latter formula is:


$$\begin{array}{l}
 \forall x, y, z, u \\
 ((\text{gcd}(x, y, z) \leftarrow \text{gcd}(y, u, z) \wedge \text{rest}(x, y, u) \wedge x \geq y) \wedge \\
 (\text{gcd}(x, y, z) \leftarrow \text{gcd}(y, x, z) \wedge x < y) \wedge \\
 \text{gcd}(x, 0, x) \wedge \\
 \cancel{\forall z \neg \text{gcd}(6, 8, z)}).
 \end{array}$$

Note that in this example and, in fact, in many practical examples of the form " $\forall v_1 \dots f_1 \wedge \forall v_1 \dots f_2 \dots \rightarrow \neg \exists v f$ " the skolemization can be obtained in few steps. With some practice it is often even possible to formulate the problem immediately in clausal form.

Note also that in the example all clauses have the special form

$$f \leftarrow f_1 \wedge \dots \wedge f_m,$$

or, using only  $\vee$  and  $\neg$ ,

$$f \vee \neg f_1 \vee \dots \vee \neg f_m,$$


i. e. in each clause only one "positive" literal occurs. Such clauses are called *Horn clauses*.

The proof of inconsistency of sets of Horn clauses is called "logic programming" in the more restricted, technical sense. Such proofs can be carried out by iteration of one single inference rule (the "resolution rule"), which may be viewed as a general elementary computation step.

This implies that formulating problems in Horn clause form is already "programming". For such sets of formulae, the resolution procedure is an "interpreter" for the language of Horn clause programs ("PROLOG").

### 5.2.3 Resolution Theorem Proving

**Example 5.6** In the example of the previous subsection, a proof on paper can be organized as follows: We start from

$$\begin{aligned}
\text{gcd}(x, y, z) &\leftarrow \text{gcd}(y, u, z) \wedge \text{rest}(x, y, u) \wedge x \geq y, && (\text{case } x \geq y) \\
\text{gcd}(x, y, z) &\leftarrow \text{gcd}(y, x, z) \wedge x < y, && (\text{case } x < y) \\
\text{gcd}(x, 0, x) &\leftarrow, && (\text{case } y = 0) \\
&\leftarrow \text{gcd}(6, 8, z).
\end{aligned}$$

("gcd(x, 0, x) ←" is a near-at-head reformulation of "gcd(x, 0, x)" and "←gcd(6, 8, z)" is a reformulation of "←gcd(6, 8, z)". Also, for the sake of brevity, we omit all universal quantifiers.)

Now, from

$$\leftarrow \text{gcd}(6, 8, z)$$

and

$$\text{gcd}(6, 8, z) \leftarrow \text{gcd}(8, 6, z) \wedge 6 < 8 \quad (\text{case } 6 < 8)$$

and

$$6 < 8,$$

which we assume to be "built-in" knowledge in our "knowledge base", we obtain

$$\leftarrow \text{gcd}(8, 6, z).$$

From the last formula and

$$\text{gcd}(8, 6, z) \leftarrow \text{gcd}(6, u, z) \wedge \text{rest}(8, 6, u) \wedge 8 \geq 6, \quad (\text{case } 8 \geq 6)$$

and the built-in knowledge

$$\begin{aligned}
&\text{rest}(8, 6, 2) \text{ and} \\
&8 \geq 6
\end{aligned}$$

we obtain

$$\leftarrow \text{gcd}(6, 2, z).$$

From the last formula and

$$\text{gcd}(6, 2, z) \leftarrow \text{gcd}(2, u', z) \wedge \text{rest}(6, 2, u') \wedge 6 \geq 2 \quad (\text{case } 6 \geq 2)$$

and the built-in knowledge

$$\text{rest}(6, 2, 0) \text{ and} \\ 6 \geq 2$$

we obtain

$$\leftarrow \text{gcd}(2, 0, z).$$

(Note that in this step we had to introduce a new variable  $u'$ . Why? Hint: The use of  $u$  would imply that the same number  $u$  would satisfy  $\text{rest}(8, 6, u)$  and  $\text{rest}(6, 2, u)$ .)

From the last formula and

$$\text{gcd}(2, 0, 2) \leftarrow \quad (\text{case } y = 0)$$

we obtain a contradiction.  $\square$

The individual steps of the above proof procedure can be read in two different ways:

1. For example, the second step could be read: "If  $\leftarrow \text{gcd}(8, 6, z)$  was true then, because of the built-in knowledge and the Horn clause  $(\text{gcd}(8, 6, z) \leftarrow \text{gcd}(6, u, z) \wedge \text{rest}(8, 6, u) \wedge 8 \geq 6)$ , also  $\leftarrow \text{gcd}(6, 2, z)$  would be true." In this way the initial assumption that no result exists is successively reduced to similar assumption about the result of simpler computations until a contradiction is reached.
2. Atomic formulae at the right-hand side of the " $\leftarrow$ " in a Horn clause are considered as "goals" that should be satisfied for achieving the goal at the left-hand side. For example, " $\leftarrow \text{gcd}(8, 6, z)$ " can be read as the goal to find a  $z$  such that  $\text{gcd}(8, 6, z)$ . Now, a rule like  $(\text{gcd}(8, 6, z) \leftarrow \text{gcd}(6, u, z) \wedge \text{rest}(8, 6, u) \wedge 8 \geq 6)$  shows how the goal " $\leftarrow \text{gcd}(8, 6, z)$ " can be reduced to the three simpler goals on the right-hand side of the arrow. Two of these goals can be satisfied by built-in knowledge. The third goal " $\leftarrow \text{gcd}(6, 2, z)$ " remains as the goal to be achieved next.

Note that the above proof not only establishes the validity of  $\exists z \text{gcd}(8, 6, z)$  under the hypotheses (case  $x \geq y$ ), (case  $x < y$ ), and (case  $y = 0$ ) but also yields ("computes") an example of a suitable  $z$ . By tracing the proof backward, in our example it can be read off that  $z = 2$  is suitable for the last step, the last but one step etc. Hence, the answer is  $\text{gcd}(6, 8, 2)$ .

When we analyze the individual proof steps we see that the same proof technique is used in each step:

1. An atomic formula (a "goal") on the right-hand side of a Horn clause is "unified" (i. e. "made equal" by a suitable substitution) with the atomic formula on the left-hand side of some other Horn clause.
2. The unified atomic formulae are canceled and the remaining atomic formulae are collected in a new Horn clause.

Amazingly, this simple proof technique (which needs a careful definition of the "unification" process) is so strong that it suffices as the only inference rule for creating a complete proof system for all of formulae in clausal form and not only for Horn clause logic. Since every formula of predicate logic can be transformed into clausal form, the resolution proof procedure is complete for all of predicate logic. The proof of this fact is not easy. It is an alternative to Goedel's proof of the completeness theorem. The details are given in the course "Automated Theorem Proving".

We now present the resolution proof procedure in its general form.

#### Resolution Proof Procedure:

Given:  $CF$ , a formula in clausal form.

Question: Is  $CF$  satisfiable?

Procedure:

$CF' := CF$

while exists  $R$

such that  $R$  is a resolvent of clauses in  $CF'$  and  $R \notin CF'$

do

$CF' := CF' \cup \{R \mid R \text{ is a resolvent of clauses in } CF'\}$

if  $\square \in CF'$  then return "UNSATISFIABLE"

return "SATISFIABLE". □

If the input formula  $CF$  is unsatisfiable then the above procedure stops after finitely many steps with the answer "UNSATISFIABLE". Otherwise the procedure may stop with the answer "SATISFIABLE" or it may go on forever. This behavior cannot be improved, i.e. it is inherently impossible to establish a proof procedure that would terminate also for all satisfiable input formulae (Church's theorem on the undecidability of first order predicate logic).

Alonzo Church 1936

In the above procedure we needed some auxiliary notions, which are given in the subsequent definitions. (In fact, we only give the definition of a "binary" resolvent. The general notion of a resolvent needs some additional technical details.)

**Definition 5.7 (Resolvents)** Let  $C_1$  and  $C_2$  be two clauses that have no variables in common.  $R$  is a binary resolvent of  $C_1$  and  $C_2$  iff

$$\begin{aligned} C_1 &\text{ has the form } \{l_1\} \cup D_1, \\ C_2 &\text{ has the form } \{\neg l_2\} \cup D_2, \\ &\text{where } l_1, l_2 \text{ are literals,} \\ R &= (D_1 \cup D_2)\sigma, \\ &\text{where } \sigma \text{ is a most general unifier of } l_1 \text{ and } l_2. \\ &\text{(Formally, } R = \square \text{ if } D_1 \cup D_2 = \emptyset.\text{)} \end{aligned}$$

**Definition 5.8 (Most general unifiers)**

$\sigma$  is a substitution iff

$$\sigma : V \rightarrow \text{Terms.}$$

A substitution  $\sigma$  is a unifier for the literals  $l_1$  and  $l_2$  iff

$$l_1 \circ \sigma = l_2 \circ \sigma.$$

A substitution  $\sigma$  is a most general unifier of  $l_1$  and  $l_2$  iff

$\sigma$  is a unifier of  $l_1$  and  $l_2$  and

for all unifiers  $\tau$  of  $l_1$  and  $l_2$

there exists a substitution  $\theta$  such that  $\tau = \sigma \circ \theta$ .  $\square$

In the above definition,  $l \circ \sigma$  denotes the application of the substitution  $\sigma$  to the literal  $l$ . (The formal definition of  $\circ$  would need some technicalities).

We now provide some simple examples that illustrate the execution of the resolution proof procedure:

**Example 5.9**

Set of clauses:

$$(1) \quad \neg C(x) \vee R(x)$$

$$(2) \quad C(a)$$

$$(3) \quad P(a)$$

$$(4) \quad \neg P(x) \vee \neg R(x)$$

Resolution procedure:

$$(5) \quad R(a) \quad \text{from (1) and (2)}$$

$$(6) \quad \neg R(a) \quad \text{from (3) and (4)}$$

$$(7) \quad \square \quad \text{from (5) and (6).}$$

Hence, the set of clauses is unsatisfiable.

### Example 5.10

Set of clauses:

- (1)  $\neg E(x) \vee V(x) \vee S(x, f(x))$
- (2)  $\neg E(x) \vee V(x) \vee C(f(x))$
- (3)  $P(a)$
- (4)  $E(a)$
- (5)  $\neg S(a, y) \vee P(y)$
- (6)  $\neg P(x) \vee \neg V(x)$
- (7)  $\neg P(x) \vee \neg C(x)$

Resolution procedure:

- (8)  $\neg V(a)$  from (3) and (6)
- (9)  $V(a) \vee C(f(a))$  from (2) and (4)
- (10)  $C(f(a))$  from (8) and (9)
- (11)  $V(a) \vee S(a, f(a))$  from (1) and (4)
- (12)  $S(a, f(a))$  from (8) and (11)
- (13)  $P(f(a))$  from (5) and (12)
- (14)  $\neg C(f(a))$  from (7) and (13)
- (15)  $\square$  from (10) and (14)

Hence, the set of clauses is unsatisfiable.

### Example 5.11

Set of clauses:

- (1)  $P_1(a) \vee P_2(b)$
- (2)  $\neg P_2(b)$
- (3)  $\neg P_1(a) \vee P_2(b) \vee \neg P_3(a)$

Resolution procedure:

- (4)  $P_1(a)$  from (1) and (2)
- (5)  $P_2(b) \vee \neg P_3(a)$  from (1) and (3)
- (6)  $\neg P_1(a) \vee \neg P_3(a)$  from (2) and (3)
- (7)  $\neg P_3(a)$  from (2) and (5)

No more resolvents can be generated in this stage. Hence, the set of clauses is satisfiable.

*Construction of a satisfiable interpretation:*

- (1)  $\Rightarrow P_1(a) = F$
- (2)  $\Rightarrow P_2(b) = T$
- (3)  $\Rightarrow P_3(a) = F$