Preface

This book started as notes for a postgraduate course in Mathematical Reasoning given in the Department of Artificial Intelligence at Edinburgh from 1979 onwards. Students on the course are drawn from a wide range of backgrounds: Psychology, Computer Science, Mathematics, Education, etc. The first draft of the notes was written during a terms sabbatical leave in 1980. Later they were used for a similar course at undergraduate level.

While there are now several textbooks on Artificial Intelligence techniques and, more particularly, on Problem Solving and Theorem Proving, I felt the need for a book concentrating on applications of these techniques to Mathematics. There was certainly enough material, but it was scattered in research journals, conference proceedings and theses. If it were collected together I hoped it might prove of interest to a wider audience than the usual artificial intelligentsia; I hoped that mathematicians and educationalists might find it a eye opener to how computational ideas could shed light on the process of *doing* Mathematics.

I also hoped to give more unity to some, rather disparate, pieces of research. In particular, I wanted to show how the, so called, 'non-Resolution theorem proving' techniques could be readily brought into a Resolution framework, and how this helped us to relate the various techniques – creating coherence from confusion. In order to achieve this goal I have taken strong historical liberties in my descriptions of the work of Boyer and Moore, Gelernter, Lenat, etc. I have redescribed their work in a uniform framework, ignoring aspects of peripheral interest, and focussing on what I take to be their essential contribution. I call such descriptions, *rational reconstructions*. This does not imply that the original work was irrational – only that my reconstructions are rational. I apologise to any of the rationally reconstructed who feel mistreated. My excuse is that the reworking of research into a coherent whole is a vital, but neglected, part of Artificial Intelligence research, and that it is better to have tried and failed than never to have tried at all.

Reading Strategies

It is not necessary to be a professional mathematician or computer scientist to read this book, but the book does presuppose some mathematical knowledge. For instance, it is necessary to know what a set and a group are. I have endeavoured to make it fairly self contained, e.g. by including an introduction to mathematical logic. But self-containedness brings its own problems; if the book is not to be too long, then the pace must not be too slow. I have tried to get the reader quickly to the heart of the book – the techniques of automatic inference – without losing him on the way.

- Chapter 1 is a general introduction; it motivates the subject and gives some of the historical background.
- Part I is a three chapter introduction to Mathematical Logic; it describes only those aspects of logic that are required to understand the rest of the book, and may be omitted by anyone who understands elementary predicate logic.
- Part II is a three chapter introduction to Resolution theorem proving. It may be omitted by anyone who knows what SL Resolution is.
- Part III consists of five rational reconstructions of theorem proving techniques or programs. Each was selected because it contributes an important partial solution to the problem of guiding the search for a proof. This part is the heart of the book.
- Part IV is a two chapter discussion of aspects of mathematical reasoning other than proving theorems – although they both reduce to theorem proving in the end.
- Part V is a three chapter investigation of the more mathematical aspects of theorem proving, e.g. completeness proofs. It may have to be omitted by those without a good background in Mathematics.
- The last chapter discusses some applications of the techniques described in the book, from algebraic manipulation to education.
- The appendices include: computer programs, notational discussions and solutions to exercises.

Scattered throughout the book are exercises of an elementary nature. Readers may want to use these to test their understanding of the text. Some of the exercises contain material that is drawn on later in the book. Solutions may be found in appendix D.

Readers already familiar with the literature of mathematical reasoning may be particularly interested in the non-standard presentations of Gelernter's Geometry Machine, in chapter 10, and Lenat's AM, in chapter 13. Section 10.5 onwards, of chapter 10, contains new results.

Acknowledgements

I would like to thank: Gordon Plotkin, who was an untiring source of information; Richard O'Keefe, Leon Sterling, Alan Borning, David Plummer, Roy Dyckhoff, Lincoln Wallen and Alberto Peterossi, who helped me debug the drafts; Bob Boyer, J Moore, Gerard Huet, Woody Bledsoe, Robert Shostak and Doug Lenat, who kindly read the chapters describing their work and gave me invaluable feedback; Alan Black, Mike Howry, Bill Clocksin, Jane Hesketh and Keh. Jiann Chen for giving feedback on the first edition of the book; and Liam Lynch and Roberto Desimone for assisting with the second edition. The students of the Department of Artificial Intelligence at Edinburgh were involuntary guinea pigs.

Contents

Chapter 1

Introduction

- Sections 1.1 and 1.2 provide motivation for studying mathematical reasoning.
- Section 1.3 gives the background of the field, describing its origins in Logic (section 1.3.1) and Psychology (section 1.3.2), and giving a short history of the automation of inference (section 1.3.3).

1.1 Why read this book?

This book is aimed at people interested in the question

How do you do Mathematics?

i.e. at professional mathematicians, students of Mathematics, teachers of Mathematics, psychologists studying mathematical reasoning and anyone else who is curious about the apparently mysterious processes by which mathematicians: conjecture theorems, formulate definitions, construct proofs and build mathematical models. My theme is that light can be shed on these 'mysterious' processes with the aid of a wonderful tool – the digital computer. By building computer programs which 'do' mathematics we can explore how it is possible to do mathematics; what the vital are talents that separate success from failure and how we all can learn to be better mathematicians.

The building of computer programs for doing mathematics is part of the new science of Artificial Intelligence. The aim of Artificial Intelligence is to study all aspects of intelligence by 'computational modelling', mathematical reasoning being just one such aspect of intelligence. Other aspects which are studied include: the ability to coordinate hand and eye to manipulate objects; the ability to hold a conversation in a, so called, 'natural' language like English (As opposed to an artificial programming language, like ALGOL or FORTRAN) or the ability to diagnose an illness and prescribe a cure.

Whatever aspect of intelligence you attempt to model in a computer program - the stacking of bricks, a cocktail party conversation or the proving of the compactness theorem - the same needs arise over and over again.

- The need to have knowledge about the domain
- The need to reason with that knowledge.

- The need for knowledge about how to direct or guide that reasoning.

Mathematical reasoning is a particularly convenient domain for studying intelligence, because the knowledge required is neatly circumscribed and the goals clear and unambiguous.

1.2 What good is Automatic Mathematical Reasoning

If we are successful in building artificial mathematicians, computer programs which do mathematics, what practical benefits will this bring?

The professional mathematician and scientist can expect a range of 'mathematical aids': programs which help him with the more tedious or complex parts of his work, programs which check his proofs, programs which offer suggestions about what to do next. Simple versions of such programs already exist. Many scientists have access to algebraic manipulation systems which can integrate symbolic expressions, simplify formulae and solve equations. We will have more to say about such programs in chapters 12 and 18.

The teacher and student of Mathematics can expect a range of computational theories which will lay bare the more mysterious aspects of the mathematician's art: how a good proof step is chosen from among the possible ones, how interesting conjectures are made, how a mathematical model is made. The teacher may also be interested in models of poor students which explain what it is that a student is doing wrong every time he gets a wrong answer. Such 'diagnostic models' will enable the teacher to design remedial instruction, tailor made, to put the student on the right road again. Again, simple versions of such theories already exist. Some primary school teachers have been given access to a program which simulates a wide range of commonly occurring, faulty, subtraction procedures. The teachers gain experience in diagnosing the fault and learn in few minutes what might otherwise take years: that most arithmetic errors are not due to 'carelessness', but are the result of carefully following a, slightly faulty, arithmetic procedure. We will say more about such programs in chapter 18.

1.3 The Historical Perspective

Of course, workers in Artificial Intelligence are not the first people to have asked how Mathematics is done. Mathematicians themselves have devoted a lot of thought to just this question. Before we start to get involved in the world of computer programs it will behove us to look at the answers they provide, see how adequate they are and how useful for our purposes.

This precomputational work on mathematical reasoning divides into two camps: the normative work of Mathematical Logic, which asks

> What are the legal modes of reasoning?

and tries to build a calculus of reasoning, and the psychological work of people who, looking at themselves and others, ask

> How do we actually go about doing mathematics?

We will see that both camps have something to offer us. Mathematical Logic will give a start on how to represent the knowledge of Mathematics and how to reason with it. It will not help us to guide that reasoning process, i.e. to decide what to do when. For that help we will turn to the psychological studies, but we will not find our needs satisfied in the kind of detail we would like. Nobody has yet succeeded in giving a precise recipe for how to be a successful mathematician. Instead we will find some good advice and useful observations and hints. These will furnish a starting point.

Let us start by looking at the work in Logic. Space does not allow more than the most superficial survey. In the succeeding chapters we will go a little deeper, but we will only be studying concepts on a 'need to know basis', so the reader wanting to know more should refer to one of the many excellent introductory books on Mathematical Logic, e.g. [47].

1.3.1 Mathematical Logic

This story, as so many stories in Science, starts in Ancient Greece. Aristotle was the first to try to describe the laws of Logic, i.e. the forms that correct arguments could take. He discovered 19 correct forms, and called them *syllogisms*. Here is an example of a syllogism

$$\frac{\begin{array}{l}\text{All Ancient Greeks were perfect.}\\ \text{Aristotle was an Ancient Greek}\end{array}}{\text{Therefore, Aristotle was perfect.}} \qquad \text{(i)}$$

The two sentences above the line are called the *hypothesis* and the sentence below the line is the *conclusion*. If the hypothesis is true then the conclusion follows from it. Notice that this syllogism is a correct argument form, regardless of whether the hypothesis is true, e.g. whether all Ancient Greeks were perfect or Aristotle was an Ancient Greek. Logicians do not worry about the *content* of arguments - whether the hypothesis is true – but only with the *form* – what sorts of argument are correct.

The form of (i) above is

$$\frac{\begin{array}{l}\text{All Ps were Q.}\\ \text{X was a P.}\end{array}}{\text{Therefore, X was Q.}}$$

The scholastics who studied and refined Aristotle's work, in the middle ages, gave names to all the valid syllogisms. The one above is called *Darii*. In our example, P was 'Ancient Greek', Q was 'perfect' and X was 'Aristotle'. We could substitute anything else for P, Q and X and still have a correct form, e.g. 'syllogism' for P, 'invalid' for 'Q' and 'Darii' for X. We can also change the tense from past to present.

$$\frac{\begin{array}{l}\text{All syllogisms are invalid.}\\ \text{Darii is a syllogism.}\end{array}}{\text{Therefore, Darii is invalid.}}$$

All the sentences involved in the 19 syllogisms, whether as hypothesis or conclusion, take one of a small number of fixed forms, like 'All As were B.' or 'Some

As are B.', so it is not possible to capture with the syllogisms alone, reasoning which involves sentences in other forms.

Thus Aristotle's list of 19 syllogisms do not exhaust the correct argument forms. Unfortunately, due to Aristotle's prestige, it was believed for many years that they did. The scholastics believed in the content, rather than just the form, of syllogism (i) above. This misplaced faith held up the development of logic for about a millennium and it took a considerable effort of will by Boole in the 19th century to add to the accepted correct forms. Here is one of Boole's additions.

Either all reasoning is syllogistic or Aristotle was wrong.
All reasoning is not syllogistic. (ii)
___
Therefore, Aristotle was wrong.

Boole invented Propositional Logic – a mathematical theory covering the way in which elementary sentences (or *propositions*) can be combined with connecting words (or *connectives*) like *and, or, not, if,* etc.

Notice how the first hypothesis of the form (ii) above consists of two propositions: 'all reasoning is syllogistic' and 'Aristotle was wrong', connected together with the connective, 'Either ... or ...'. In fact the form of (ii) is:

Either P or Q.
Not P.
___
Therefore, Q.

Boole's Propositional Logic and Aristotle's syllogisms described two disjoint families of correct argument forms, but they still did not exhaust all the correct forms. Both families are too inflexible, but in different ways. The syllogisms only allow reasoning between sentences of a few simple forms. We cannot connect together several sentences to form a hypothesis in the way that we can in Propositional Logic. Propositional Logic, on the other hand, can be used to build hypotheses and conclusions of arbitrary complexity from propositions, but we cannot delve inside an proposition itself as we can in a syllogism, e.g. we cannot extract the bits, 'reasoning' and 'syllogistic'

from 'All reasoning is syllogistic.'.

The liberation of the correct argument forms was provided by Gottlieb Frege, who invented Predicate Logic. Here the basic building blocks are *objects* and relations (or *predicates*) between them. Predicate Logic combines Propositional Logic's ability to construct new sentences from old, with the syllogistic ability to delve into the internal structure of the sentences. It includes all the correct forms of both its predecessors, and more besides. Here is one of the new forms

This is an argument.
This is not propositional.
This is not syllogistic.
___
Therefore, some arguments are neither propositional nor syllogistic.

Notice how properties of being an argument, a proposition and syllogistic are extracted from inside the three hypothesis sentences and used to construct a new sentence, together with the *quantifier*, some, and the connective, neither ... nor ... .

Predicate Logic includes nearly all the correct argument forms we shall have cause to use in this book, although we will occasionally stray into higher order logics. We will be taking it up again in chapters 2 to 4, where we will use it to represent mathematical knowledge and mathematical reasoning. Our justification will be a theorem, due to Jacques Herbrand, which suggests an automatic procedure for finding proofs to theorems, which is guaranteed to find a proof if there is one. The procedure suggested by Herbrand's theorem turns out to be horribly inefficient, but it will serve as a starting point.

1.3.2 Psychological Studies

Mathematical Logic concerns itself with *justification* rather than *discovery*. That is, we can use its result to show that an existing mathematical proof is correct, but despite Herbrand's Theorem, it is not a lot of use in actually finding the proof in the first place. Finding a proof (or even deciding what theorem to try and prove) is largely a matter of 'experience', 'luck', 'intuition' and all the other mysterious processes which we sometimes feel are beyond understanding. The purpose of this book is to establish that, on the contrary, these processes can be understood, and even modelled in a computer program. To help establish this, we will now appeal to those few mathematicians who have tried to capture something of their skill in words.

The most famous of these is George Polya. In his book, 'How to Solve It' [54], he summarized and explained some advice, designed to improve the reader's problem solving ability. This advice took the form of questions for the problem solver to ask himself.

> What is the unknown?
> Do you know a related problem?
> Could you restate the problem?

Generations of students have found these questions an inspiration to their problem solving efforts. They undoubtably help to liberate thought by suggesting new avenues and pointing to the continuation of old ones. How helpful are they to us in building an artificial mathematician?

Unfortunately, Polya's exhortations lack the detail required to make them immediately useful in our task. Consider, for instance, 'Could you restate the problem?'. This question presumes knowledge of English and the way in which problems can be stated in it, which we are only just beginning to give our computers. To be able to make use of a restatement of the problem implies knowing subtle relationships between problem statements and solution methods which our computers just do not know. We must start by building up an armoury of solution methods and relating them to problem types. Then we might be able to build a computer program which could understand Polya's advice – a first step on the road to using it.

All is not lost, however. A later book of Polya's, 'Mathematical Discovery' [55], holds out more hope. The advice given in this two volume set is much more domain specific. For instance, the first chapter of the first volume gives specific help on making geometric constructions. This advice *has* been embodied in a computer program [23]. Polya's attitude in trying to understand the 'mysterious' aspects of problem solving is all too rare. The usual attitude of mathematicians is reflected in their published research papers and in mathematics textbooks. Proofs are revamped and polished until all trace of how they were discovered is completely hidden. The reader is left to assume that the proof came to its originator in a blinding flash, since it contains steps which no one could possibly have guessed would succeed. The painstaking process of trial and error, revision and adjustment are all invisible.

The only attempt, of which I am aware, to explain the process by which a proof was constructed, is B.L van der Waerden's paper, 'How the proof of Baudet's conjecture was found', [22]. Here is an excellent description of the process of trial and error, an induction hypothesis is proposed, and gradually modified, as successive attempts to prove the induction step fail. Yet each failure suggests the modifications which take the next attempt a step further.

Imre Lakatos undertook a more ambitious task – to chart the development of a particular theorem through several centuries [38]. He describes the trial and error processes which have given us our modern version of Euler's Theorem, that

> for all polyhedra, $V-E+F=2$, where V is the number of vertices, E the number of edges and F the number of faces.

The conjecture of the theorem from a number of examples, the derivation of a proof, the discovery of a counterexample, the location of the fault in the proof, the revision of the proof and the many forms this revision may take. The theme of his book is the subtle interaction between proof and counterexample: how a flawed proof may suggest a counterexample, how a counterexample may suggest ways to improve a proof. For instance, we may change the definitions of key concepts in the theorem, like the concept of 'polyhedra', in order to exclude the counterexample.

A major implication of Lakatos's discussion, is that the notion of *definition* itself is a subtle one. Not only do different people at different times have different definitions of concepts like polyhedra, but the definition of a single person at a single time may be hazy, so that it cannot be used to decide whether some difficult 'borderline' case is a polyhedron or not. How useful are the observations of van der Waerden and Lakatos in our task of building an artificial mathematician? Again, their advice, while extremely important, is premature given our current state of development. Workers in Artificial Intelligence have been struggling to build programs to generate sensible first proof attempts, which could form the basis for such refinement processes. Until they succeed at this, the observations of van der Waerden and Lakatos will go unused.

However, we will see in chapters 10 and 11 that there have been some attempts to guide the search for a proof both by the use of counterexamples and by gathering evidence from an earlier failed attempt at a proof, although neither of these computer programs displays the sophistication observed by van der Waerden and Lakatos.

1.3.3 Automatic Theorem Proving

The idea of building an 'artificial mathematician' can be traced back to two sources: the invention of the digital computer and the development of Mathematical Logic. This was compounded by the fact that some very able mathematicians, men like Alan Turing, were engaged in both enterprises. Unfortunately, the psychological observations described above played a very minor role in the early days.

Mathematical Logic provides a formal theory of Mathematics. That is, it shows how any branch of Mathematics can be described as the derivation of theorems from a set of axioms using some rules of inference. These rules of inference are just a selection of the correct argument forms which we illustrated in section 1.3.1 above. The axioms are some sentences which we decide to accept as true and which can then be used as the hypothesis of a rule of inference to derive the conclusion of the rule as a theorem. The classic model for this is Euclidean Geometry.

This suggests a simple procedure for developing a mathematical theory. Starting with just the axioms as the 'theorems' of the theory we can pick a rule of inference at random, find some theorems which can be used as its hypothesis and add its conclusion to our pool of theorems. Thus if we pick as our rule of inference the Darii syllogism

$$\frac{\begin{array}{l}\text{All Ps are Q.}\\ \text{X is a P.}\end{array}}{\text{Therefore, X is Q.}}$$

and we already have in our pool of theorems

All odd numbers are prime
and
9 is an odd number

then we could substitute 'odd number' for P, 'prime' for Q and 9 for X and derive the new theorem

9 is prime

Of course, this would be a funny mathematical theory, but we should be able to model correct reasoning from faulty assumptions as well as from correct assumptions.

If we were interested in proving a particular conjecture we could just go on generating new theorems hoping that the one we wanted would turn up. This would be a hopeless method for humans to use. All sorts of irrelevant, and probably uninteresting, theorems would get generated before our conjecture was derived. Even if our conjecture were provable there is no knowing when it might get proved. However, digital computers are known for their speed and their capacity to do oodles of boring work without making mistakes. This procedure might be a practical one for them.

Well it isn't. Despite their impressive speed, the number of possible moves in this mathematical game are too great. Using this technique, a computer can be made to churn out trivial and uninteresting 'theorems' at an enormous rate, but unless the one you are interested in has a particularly simple proof it is unlikely to turn up this side of doomsday.

To see why this is so, consider the 'substitution' rule of inference, which most mathematical theories contain.

$$\frac{A(X)}{A(T)}$$

This is to be read as follows

> If a theorem A contains a variable X then we may substitute any term T for X in A to form a new theorem, $A(T)$.

The catch here is the 'any term T'. If this is a theory about arithmetic then 'any term' might mean any natural number, 0,1,2,3,... etc. and any combination of these with arithmetic operations, $+$, $\cdot$, $/$ etc. and even other variables. So from the theorem $X \neq X+1$, we may generate $0 \neq 0+1$, $1 \neq 1+1$, $1+2 \neq 1+2+1$, $1+Y \neq 1+Y+1$, and so on. This is an awful lot of new theorems – an infinite number – and worse still those new theorems which contain variables can now be used to generate new ones in their turn.

This is where Herbrand's Theorem comes to the rescue. The effect of his theorem, which is described in more detail in appendix 16, is to limit the 'any terms' that we need consider substituting for X. We can restrict ourselves to terms without variables, and we need only consider those which can be constructed from symbols already occurring in the conjecture and the axioms. It also suggests adding the negation of the conjecture as a new axiom and searching for a contradiction, rather than searching for the conjecture among those generated.

The theorem proving process implied by Herbrand's Theorem was implemented by Paul Gilmore in 1960 [26]. It turned out to be pretty inefficient. It could generate the proofs of a few trivial theorems, but got hopelessly lost when trying to prove anything slightly more complicated.

But Gilmore's program showed that automatic theorem proving was possible in principle. During the rest of the decade there was a concerted attempt to improve the basic . The key event in this period was Alan Robinson's invention of the *Resolution* procedure [59]. Robinson showed how the standard axioms and rules of inference of Predicate Logic could be replaced with the single, slightly complicated, rule he called resolution. Naturally this rule included elements of the explosive, substitution rule, but in a way made subservient to the context: the only substitutions attempted were those which enabled other rules to be applied. Appropriate substitutions were calculated by a procedure called *unification*. We will study unification and resolution more in parts II and V.

For the remainder of the decade nearly all automatic theorem proving systems were either refinements or extensions of the Resolution procedure. Systems were built with wonderful sounding names, like Hyperresolution, Paramodulation, P1 Deduction, SL Resolution, Lush Resolution, Connection Graphs. Each of them represented an improvement over the original Resolution procedure and steady progress was made, with ever more difficult theorems being proved. There seemed no reason why this situation could not continue indefinitely, until the whole of Mathematics was gradually conquered.

The tranquility was shattered by strident criticism from other workers in Artificial Intelligence, especially those from the Massachusetts Institute of Technology. They argued that this work, on what they termed 'Uniform Proof Procedures', was not going to conquer the whole of Mathematics, but only a trivially small subset of Mathematics. To generate proofs of interesting, non-trivial theorems required the use of sophisticated, domain specific knowledge, for which no provision was made in the Resolution family of theorem provers.

This attack was at first resisted and then reluctantly accepted. The development of Resolution theorem provers ceased, except in a few isolated pockets of resistance. People began to look at the works of Polya and Lakatos and to introspect about their own mathematical activity in order to get inspiration as to how to proceed. A new family of *Natural Deduction* theorem provers emerged. All sorts of new techniques were attempted: domain specific guidance, the use of models and counterexamples, rewrite rules, analogical reasoning. A main goal of this book is to try to explain the work done in this period and to give order to it.

1.4 Summary

This chapter has introduced the building of computer programs to do mathematical reasoning. We have motivated the building of 'Artificial Mathematicians' and described some of the historical background to the attempt to do so. In subsequent chapters we will consider *how* to do it.

Part I

Formal Notation

Chapter 2

Arguments about Propositions

- This chapter is an introduction to Propositional Logic.
- Section 2.1 introduces the truth functional connectives.
- Section 2.2 considers formulae made from connectives and propositions. It defines the meaning of such formulae using semantic trees (section 2.2.1) and shows how to use these to identify correct argument forms (sections 2.2.3 and 2.2.4).

Although those involved in the renaissance of systems and techniques developed in the 1970s were not always aware of their debt, all of them relied heavily on the 1960s work on Resolution. To help give unity to all these later efforts we will recouch them all in the language of Predicate Logic and Resolution. This will involve us in studying sufficient of the field of Mathematical Logic to understand how mathematical knowledge and reasoning can be represented in a mathematical calculus and to understand the significance of Herbrand's Theorem and the resolution rule of inference. This chapter deals with Propositional Logic; the next chapter extends this to Predicate (or First Order) Logic; and the chapter after that extends this again to Omega Order Logic.

Readers familiar with Mathematical Logic may wish to skip now to part II and those familiar with Resolution may wish to skip beyond that to chapter 7. Let me tempt even the experts to stay by announcing that my approach to Logic is a little nonstandard, having been specially designed to support the theoretical demands of automated reasoning.

For those who are still with us we will start our story with logical notation and how it can serve, as a alternative to mathematically flavoured English, as a tool for representing mathematical statements in a form suitable for manipulation by a computer.

2.1 Truth Functional Connectives

In section 1.3.1 we mentioned the Propositional Logic s, and, or, not etc, and how they can be used to connect s together. In this section we explore these

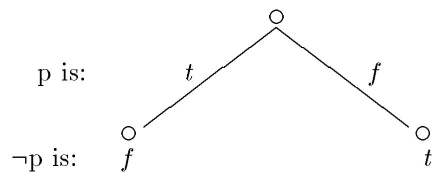


Figure 2.1: Semantic Tree for $\neg p$

connectives a little more deeply: defining them properly and illustrating them with examples drawn from Mathematics.

2.1.1 Negation

In Mathematics we have a variety of ways of saying that something is not the case. We may use English and say

n is not a prime number

or we may use some special mathematical notation, like

$a \neq 0$ or $a \not< b$

when we want to say that $a = 0$ is not true or that $a < b$ is not true.

Using English as an internal representation in a computer program brings its own problems, not least of which is the essential ambiguity of English statements. So we will want to avoid the English formulations. It will also be helpful to regularize the various mathematical conventions. We will therefore adopt the *negation* symbol $\neg$, writing $\neg p$ when we mean that statement p is not true. Thus we will write:

$\neg n$ is a prime number
$\neg a = 0$ and
$\neg a < b$

$\neg$, like the other propositional connectives, is *truth functional*, that is the truth of $\neg p$ depends only on the truth of p. $\neg p$ is false just when p is true and $\neg p$ is true just when p is false. We can sum up this relationship with a *semantic tree*, see figure 2.1. The word semantic means 'concerned with meaning'; the semantic tree is a which gives the meaning of $\neg$. For those not familiar with mathematical trees, appendix B explains the terminology. This, rather simple, tree has two s coming from the , corresponding to the two possible truth values for p, true and false (abbreviated as t and f). At the s of each branch it has the truth value of $\neg p$ corresponding to the values assigned to p on that branch. We will meet more complicated trees in the following sections.

2.1.2 Conjunction

Next we will consider how two statements can be connected so that the truth of both of them is asserted. In English, this can be done with words like 'and', 'but', 'where', etc. Consider, for instance,

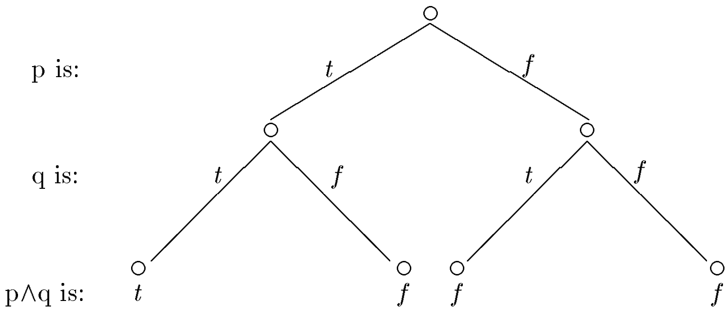


Figure 2.2: Semantic Tree for $p \wedge q$

$2 < X$ and $X < 10$
$2 < X$ but $X \neq 3$
$a \cdot X + b = 0$ where $a \neq 0$

There are also special mathematical conventions like:

$2 < X < 10$

We will replace all of these with the *conjunction* symbol, $\wedge$. That is, $p \wedge q$ will mean both p and q are true. Again we can make this rather more precise by defining $\wedge$ with a semantic tree, as in figure 2.2. The truth values given at the tips of the branches are determined by the values assigned to p and q on those branches. Thus if we start from the root and follow the arc where p is true then the arc where q is false, we find that $p \wedge q$ is false.

2.1.3 Disjunction

Just as two statements can be connected to assert the truth of both, they can also be connected to assert the truth of one or the other. This effect can be achieved in English by saying

$$X \leq Y \text{ or } X \geq Y \tag{i}$$

or we can use special mathematical conventions like

$$X = \pm 2 \tag{ii}$$

This is called *disjunction*.

As usual we will use a special symbol, $\vee$, to indicate disjunction. That is, $p \vee q$, will mean that at least one of p and q is true. We will write:

$X < Y \vee X > Y$
$X = 2 \vee X = -2$

The semantic tree for $\vee$ is shown in figure 2.1.3 Notice how we have settled the ambiguous case when both p and q are true. We have made $p \vee q$ true. This is called *inclusive or*. If we had chosen to make $p \vee q$ false we would have defined *exclusive or*. Both cases are common in Mathematics (we gave an

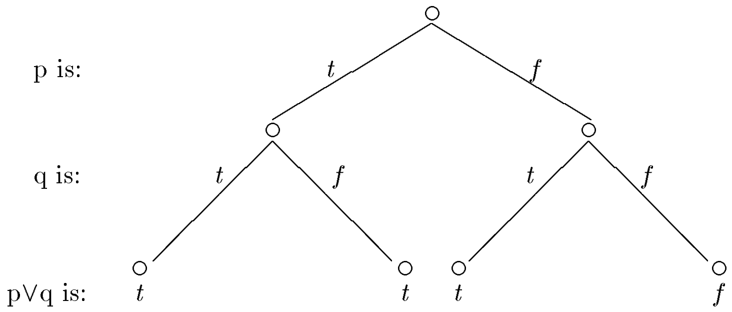


Figure 2.3: Semantic Tree for $p \vee q$

example of each case in i and ii above). Inclusive or is usually given precedence in Propositional Logic because it has a nice duality with $\wedge$. To see this exchange the t's for fs, and vice versa, at the tips of the semantic tree for $\vee$ and compare the result with the semantic tree for $\wedge$. Inclusive or is often sufficient to represent disjunctions like ii above, because the exclusivity of the cases is implied by the context, e.g. X cannot equal both -2 and $+2$, since $\neg 2 = -2$.

Exercise 1 *Draw the semantic tree for exclusive or.*

2.1.4 Implication

We often want to say that the truth of one statement implies the truth of another. For instance,

> If n is an odd number then n is prime
> n is prime if n is an odd number
> n is prime whenever n is an odd number
> For n to be prime it is sufficient for n to be odd
> n is an odd number implies n is prime

We will replace these English formulations with the *implication* arrow, $\rightarrow$, writing

> n is an odd number $\rightarrow$ n is prime

The semantic tree for implication is given in figure 2.4. Notice how we have decided the ambiguous cases where p is false, by making $p \leftarrow q$ true, regardless of the value of q. This version of implication is called, *material implication*. It is the most common version in Propositional Logic, doubtless because it bears a pretty relationship to $\vee$.

2.1.5 Double Implication

We often want to assert that not only $p \rightarrow q$ but also that $q \rightarrow p$. This is sometimes done by using the phrase 'if and only if' or its conventional shortening 'iff'. It is also done by phrases like 'a necessary and sufficient condition'.

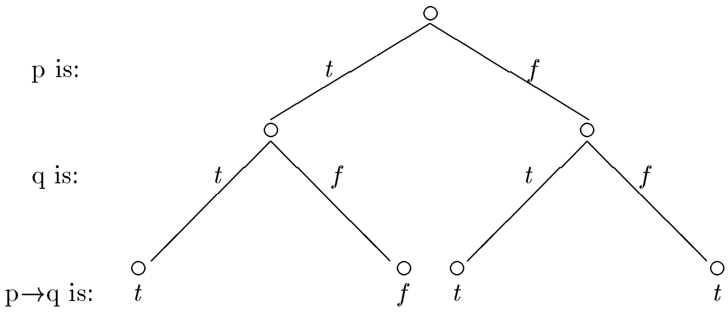


Figure 2.4: Semantic Tree for $p \to q$

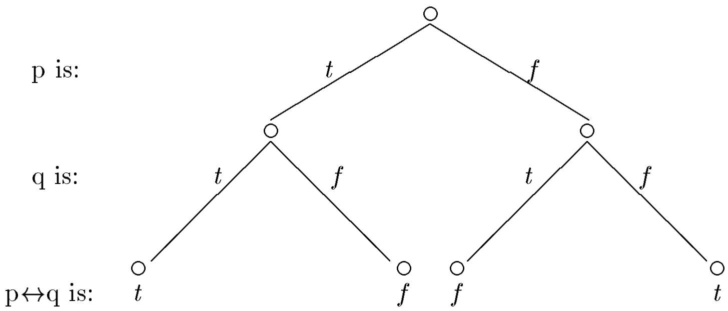


Figure 2.5: Semantic Tree for $p \leftrightarrow q$

> $X+2=3$ if and only if $X=-1$
>
> For a number to be divisible by 15 it is necessary and sufficient that it be divisible by 3 and by 5

We will replace these English formulations with *double implication* doubled headed arrow, $\leftrightarrow$.

A note of caution: The word 'if' is often used where double implication is intended, especially in definitions. Consider, for instance,

> A number is prime if it has exactly two divisors.

Presumably, this is the only way a number can be prime. The reader is meant to interpret the 'if' as 'iff'.

The semantic tree for double implication is given in figure 2.5. Notice the duality between this tree and the one for exclusive or which you drew in exercise 1.

2.2 Propositional Formulae

Now that we have some connectives: $\neg$, $\wedge$, $\vee$, $\leftarrow$, $\leftrightarrow$, we can begin to use them to put together some more complicated statements. Suppose we start with the propositions:

> n is an odd number,
> n is prime

Then we can use negation to form

> $\neg n$ is prime

and then disjunction to form:

> $\neg n$ is an odd number $\vee$ n is prime

and then conjunction to form:

> $\neg n$ is an odd number $\vee$ n is prime $\wedge$ n is an odd number

and so on, and so on.

It will help clear up ambiguities if we establish a precedence ordering among the connectives and use brackets to clear up any remaining conflicts, e.g.

> – $\neg n$ is an odd number $\vee$ n is prime″ $\wedge$
> n is an odd number

We can omit brackets around '$\neg n$ is an odd number' because the conventional precedence ordering is that $\neg$ binds tighter than the other connectives, so that $\neg p \vee q$ means $(\neg p) \vee q$ rather than $\neg(p \vee q)$. This is similar to the situation in arithmetic where $-2+3$ means $(-2)+3$ rather than $-(2+3)$.

The precedence order is that $\neg$ binds tightest, $\wedge$ and $\vee$ bind next tightest, and $\leftarrow$ and $\leftrightarrow$ bind loosest. Hence,

$\neg p \vee q$	means	$(\neg p) \vee q$
$p \wedge q \leftarrow r \vee s$	means	$(p \wedge q) \leftarrow (r \vee s)$
$p \wedge q \vee r$	is ambiguous	

We can summarize this ability to form new formulae from old in the following *recursive* definition. This definition is called *recursive* because it appeals to itself, but without getting into an infinite regression.

Definition 2.1 *Formulae*

1. *A proposition is a formula.*
2. *If p and q are formulae then the following are also formulae: $\neg p$, $p \wedge q$, $p \vee q$, $p \leftarrow q$, $p \leftrightarrow q$.*
3. *Only expressions formed by rules 1 and 2 above are formulae.*

It will be convenient to use the letters p, q, r etc, as above, to stand for arbitrary propositions, e.g. $\{\neg p \vee q\} \wedge p$.

We will sometimes want to represent formulae as s. The s of these trees will be labelled by propositions and the other s by connectives. The label of the root node is said to be the *dominant* connective. For instance, $\{\neg p \vee q\} \wedge p$ will be represented as

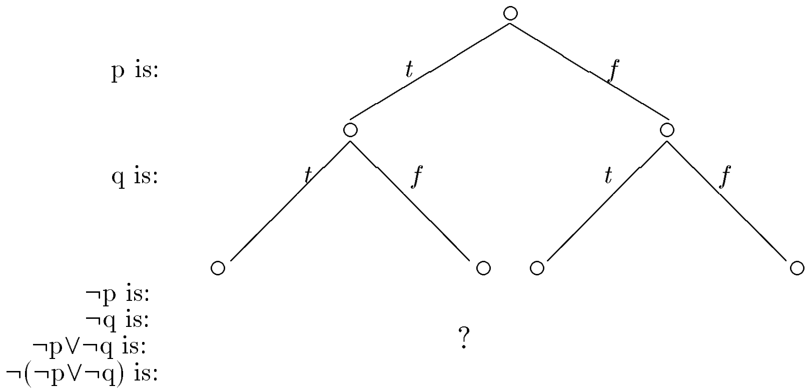
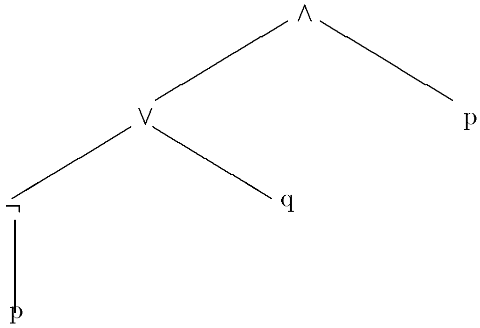


Figure 2.6: Semantic Tree Framework for Two Propositions: p and q



Note how the tree reflects the recursive structure of the formula. Each subformula is represented as a subtree. If a formula is made from a connective and some subformulae then its tree is formed from the subtrees of these subformulae connected by a node labelled by the connective.

2.2.1 Semantic Trees

We can draw semantic trees for these more complex formulae, just as we did for the simple ones in section 2.1. Consider the formula:

$$\neg(\neg p \vee \neg q)$$

The first step is to list the propositions it contains: in this case just p and q. We then build a semantic tree for these sentences, with no labels on the tips.

Below this tree we can list the subformulae, of the formula in question, in increasing order of complexity as shown in figure 2.6.

We can then fill in the values of each of these formulae, below each tip, as follows:

- Consider each of the formulae in top/bottom order.

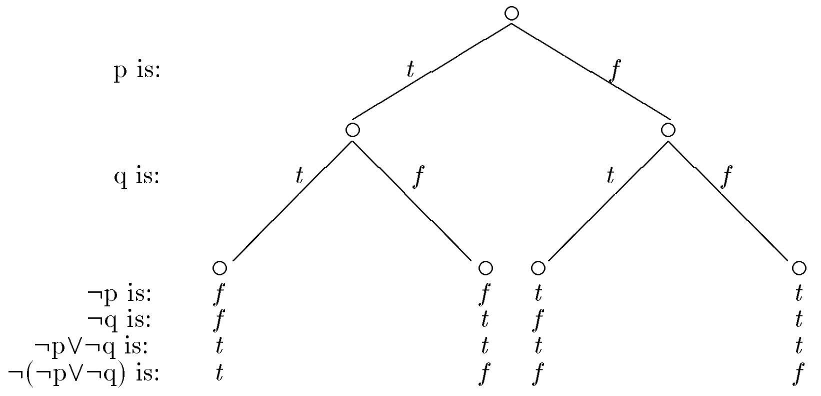


Figure 2.7: Semantic Tree for $\neg(\neg p \vee \neg q)$

- Consider the dominant connective of the formula. Its parameters will have been assigned values either on the corresponding branch of the tree or by some previous iteration of this process.
- Look up the semantic tree for this connective. Find the branch which assigns these values to its parameters. Find the value at the tip of this semantic tree. This is the value of the formula on the current tip.

Thus, suppose we were trying to fill in the value of $\neg p \vee \neg q$ below the second tip (i.e. the slot marked ? in figure 2.6). We assume that the rows for $\neg p$ and $\neg q$ have already been filled in and the values f and t, respectively assigned to the second tip. Looking at the semantic tree for $\vee$ (figure 2.1.3) we see that when f and t are assigned to the parameters of $\vee$ that t is assigned to the tip. Thus we replace ? with t.

Repeating this process for the remaining formulae and tips gives the tree in figure 2.7. Compare this tree with the tree for $p \wedge q$. It is the same. $p \wedge q$ and $\neg(\neg p \vee \neg q)$ have the same value for all assignments of truth values to p and q. We say that the two formulae are *logically equivalent* or just *equivalent* for short.

Exercise 2 *Draw a semantic tree for the expression $\neg p \vee q$. Compare this with the semantic tree for $p \rightarrow q$.*

2.2.2 Equivalences

Using semantic trees we can establish equivalences between different propositional formulae. We have already seen that $p \wedge q$ and $\neg(\neg p \vee \neg q)$ are equivalent. And when doing exercise 2 above you should have noticed that $\neg p \vee q$ and $p \rightarrow q$ were also equivalent. You may also like to show that $p \leftrightarrow q$ and $(p \rightarrow q) \wedge (q \rightarrow p)$ are also equivalent.

These discoveries should not come as too much of a surprise. Equivalent formulae 'say the same thing'. If you reflect on the meaning of $p \leftrightarrow q$ –p if and only if q″ and $(p \rightarrow q) \wedge (q \rightarrow p)$ –p implies q and q implies p″ then you will see

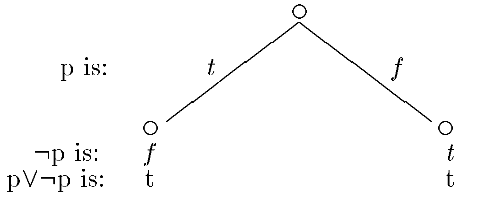


Figure 2.8: Semantic Tree for $p \vee \neg p$

that they are really two different ways of 'saying the same thing'. Try the same exercise with the other equivalent formulae above until you convince yourself that they really say the same thing too.

The discovery of these equivalences means that there is some redundancy in our connectives. We need not have introduced the connective, $\leftrightarrow$, at all. Whenever we felt the need for it we could have replaced it with the equivalent $(p \to q) \wedge (q \to p)$. However, this equivalent expression is a bit clumbersome. It is often more convenient to use the shorter, but redundant,$\leftrightarrow$.

In a similar way we could replace all occurrences of $p \wedge q$ with the equivalent $\neg[\neg p \vee \neg q]$ and all occurrences of $p \to q$ with the equivalent $\neg p \vee q$. In this way we can whittle down the connectives we actually need to two, $\vee$ and $\neg$.

In fact, if we had introduced the connectives, Sheffer stroke and dagger (also known as Nand and Nor), we would have found that either one of them would do, all on its own. Their definitions are:

$$p \mid q \leftrightarrow \neg(p \wedge q)$$
$$p \downarrow q \leftrightarrow \neg(p \vee q)$$

2.2.3 Tautologies and Contradictions

Consider the formula, $p \vee \neg p$. If we build its semantic tree we will discover that the labels of all its tips are t. That is, it is always true, regardless of the values assigned to its propositions. This is really not surprising. After all it says that either p is true or p is not true. A fairly obvious observation. Such a formula is called a *tautology*.

Exercise 3 *Show that* $\neg\neg p \leftrightarrow p$ *and* $(p \to q) \leftrightarrow (\neg p \vee q)$ *are also tautologies.*

As well as formulae whose semantic trees have t at all their tips, we can have formulae whose semantic trees have f at all their tips, i.e. formulae which are false for all assignments. Such a formula is called a *contradiction*.

Exercise 4 *Show that* $p \wedge \neg p$ *is a contradiction.*

In exercise 3 you will have shown that a tautology can be formed from a double implication between two of the formulae we showed equivalent in section 2.2.2. This is true in general. If A and B are two equivalent formulae, then $A \leftrightarrow B$ is a tautology. The reverse is also the case. If $A \leftrightarrow B$ is a tautology then A and B are equivalent.

Theorem 2.2 *A and B are equivalent formulae if and only if $A \leftrightarrow B$ is a tautology.*

Proof: If A and B are equivalent then, by definition, they have the same truth value for all assignments of truth values to the propositions they contain. Consider any tip of the semantic tree of $A \leftrightarrow B$. On the branch above this tip various assignments of truth values will have been made to the propositions in A and B. Substitute these into A and B. The resulting values of A and B will be either both t or both f. In either case the value of $A \leftrightarrow B$ will be t (see semantic tree for $\leftrightarrow$). Therefore $A \leftrightarrow B$ is a tautology.

If $A \leftrightarrow B$ is a tautology then every tip of its semantic tree is labelled t. This could only have happened if the assignment on the branch above made A and B either both t or both f (see semantic tree for $\leftrightarrow$). In either case A and B have the same value. Therefore A and B are equivalent. QED

2.2.4 Identifying Correct Arguments - Part 1

The apparatus of semantic trees can be used to identify arguments whose correctness only depends on the way the propositions in it are connected. We will call such arguments, *boolean*, after George Boole who first classified such arguments. Correct boolean arguments constitute the theorems of the mathematical theory, *Propositional Logic*. We will have no need to develop this theory, e.g. we do not need to give the axioms of Propositional Logic, since they are not required in order to understand automatic theorem proving.

For instance, consider the argument we met in section 1.3.1.

$$\begin{array}{l} \text{Either all reasoning is syllogistic or Aristotle was wrong.} \\ \text{All reasoning is not syllogistic.} \\ \hline \text{Therefore, Aristotle was wrong.} \end{array} \qquad \text{(iii)}$$

This contains only two constituent propositions, 'all reasoning is syllogistic' and 'Aristotle was wrong', connected by $\vee$, $\neg$, an implicit $\wedge$ between the two hypotheses and a $\rightarrow$ between the hypotheses and the conclusion. Hence we can formalize it as.

$$\begin{array}{l} [(\text{All reasoning is syllogistic} \vee \text{Aristotle was wrong}) \wedge \\ \quad \neg\text{All reasoning is syllogistic}] \rightarrow \text{Aristotle was wrong} \end{array} \qquad \text{(iv)}$$

The semantic tree for this formula is given in figure 2.9, from which we can see that it is a tautology, i.e. its truth is independent of the truth of its constituent propositions. Hence it is a correct argument form.

The technique exemplified above can easily be implemented in a computer program which, given a formula in logical notation, e.g. (iv) above, could extract the constituent propositions, build a semantic tree and check to see if the formula is a tautology. Such a computer program would be a *decision procedure*, that is a procedure which, applied to an argument, is guaranteed to stop after a while and say whether the argument is correct or not. Any area of Mathematics, like boolean arguments, for which there is a decision procedure, is called *decidable*. In subsequent chapters we will discover areas of Mathematics for which, not only do we not have a decision procedure, but we could not ever have one. Such areas are called *undecidable*.

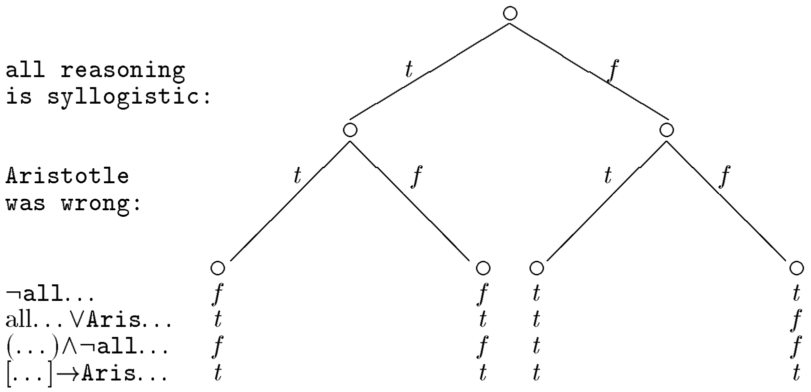


Figure 2.9: Semantic Tree for Formula (ii)

It is even possible to write a computer program to translate the mathematical English formulation of the argument, e.g. (iii), to the logical formulation, e.g. (iv), but for this we must wait until chapter 14.

Exercise 5 *Show that 'modus ponens', i.e.*

$$\frac{\begin{array}{l} P \text{ implies } Q \\ P \end{array}}{\text{Therefore } Q}$$

is a correct argument form.

How about the other argument forms in section 1.3.1? Can we identify these as correct too? Unfortunately, these rely on the internal structure of the propositions, i.e. they are non-boolean, and we will need the apparatus of the next chapter to deal with them.

2.3 Summary

In this chapter we have introduced the truth functional connectives: negation, conjunction, disjunction, implication and double implication, and defined propositional formulae, which are composed of propositions joined together by connectives. The meaning of these formulae can be calculated by constructing their semantic trees. Formulae which are true on all branches of their semantic tree are called tautologies, and those which are false on all branches are called contradictions. We can test whether a boolean argument is correct by seeing whether it translates into a tautology.

Chapter 3

The Internal Structure of Propositions

- This chapter is an introduction to Predicate Logic.
- Section 3.1 introduces the primitive parts of the proposition: the functions, predicates, variables and constants.
- Section 3.2 introduces the quantifiers.
- Section 3.3 considers formulae made from these primitive parts and defines the meaning of such formulae with the aid of interpretations.
- Section 3.4 uses interpretations to identify correct argument forms.

We have some machinery for representing the way in which propositions can be combined and a mechanical method, using semantic trees, for deciding when a formula is always true (a tautology) just by virtue of its structure and, hence, for deciding the correctness of boolean arguments. But we are still representing the propositions themselves in a mathematical English, e.g. 'n is prime'. We now turn our attention to representing the internal structure of propositions. This will lead us to extend Propositional Logic to Predicate Logic, and to a method for deciding the correctness of non-boolean arguments.

3.1 Functions and Predicates: Variables and Constants

We can draw some inspiration here from the existing practice in mathematics. A typical mathematical proposition is

$$\sin(90 - X) = \cos(X) \tag{i}$$

In this proposition we can identify several different kinds of beast.

- First there is the *constant*, 90. In this case a number which represents one of the objects about which algebraic laws, like i, above express some truth.

Arity	**Functions**	**Predicates**
0	13 e	t f
1	$\sin(X)$ $\cos(X)$	
2	$X+Y$ $X-Y$ X^Y $\log_X Y$	$X=Y$ $X<Y$ $X \mid Y$
3		$X = Y\ mod\ Z$

Table 3.1: Standard Functions and Predicates of Different Aritys

- Then there is the *variable*, X. This too represents a number, but not a fixed one like 90. X here stands for any number.
- Next are the *functions*, sin, cos and $-$. These take in numbers and return other numbers, e.g. sin takes 90 and returns 1; given 30, instead, it would return $\frac{1}{2}$.
- Lastly is the *predicate*, $=$. This is similar to the functions, but instead of taking in numbers and returning numbers, it takes in numbers and returns a truth value, true or false.

We will adopt a convention that words starting with a capital letter (sometimes just a single capital letter as here) stand for variables, whereas words starting with a lower case letter stand for constants. This will enable us to use a much wider range of symbols for variables than the normal, X, Y, Z etc.

Each proposition contains only one predicate, but it may contain any number of constants, variables and functions. Functions are applied to constants and variables to form *terms*, e.g. $90 - X$. 90 and X are called the *parameters* of $-$. These terms can then, in their turn, be used as the parameters of further functions to form new terms, e.g. $\sin(90 - X)$. Finally the terms are used as the parameters of a predicate to form a proposition, e.g. $\sin(90-X) = \cos(X)$.

Some functions and predicates take only one parameter, e.g. sin and cos. They are called *unary*. Some take two parameters, e.g. $-$ and $=$. They are called *binary*. Some take three parameters, e.g. *mod*. They are called *ternary*. More generally a function or predicate which takes n parameters is called *n-ary*. The number of parameters a function or predicate takes is called its *arity*. Unary predicates are sometimes called *properties* and non-unary predicates *relations*. *Nullary* functions and predicates are also allowed, i.e. those which take no parameters. In fact we have already met them both: The nullary functions are just the constants and the nullary predicates are the truth values, t and f.

Table 3.1 contains some standard functions and predicates of different aritys (applied to variable parameters to aid clarity).

When we want to represent an arbitrary function or predicate of arity n, applied to some parameters we will use the notation

$f(t_1, \ldots\ldots, t_n)$

and

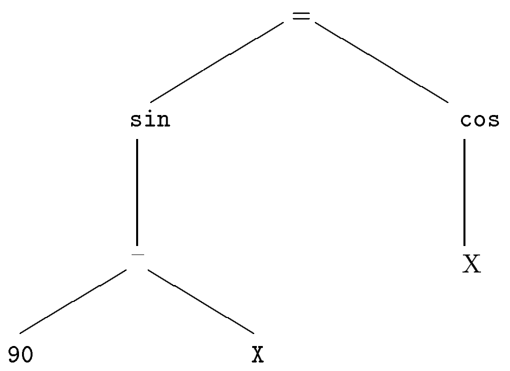


Figure 3.1: The Representation of $\sin(90 - X) = \cos(X)$
as an Expression Tree

> $p(t_1, \ldots.., t_n)$
> where the t_i are arbitrary terms.

When the t_i are of only peripheral interest we will sometimes use bold face letters to indicate a vector of n parameters, e.g. $f(\mathbf{t})$ and $p(\mathbf{t})$.

This notation, where the function or predicate symbol is followed by its parameters contained in parenthesis, is called *Functional Form*. As can be seen in table 3.1, it is rarely used in practice. Symbols are often *infixed* between the parameters, e.g. $X + Y$. Parameters appear as sub- or super-scripts to symbols and even to other parameters. No holds are barred! We will generally stick to the established mathematical practice, otherwise this book runs the risk of becoming more obscure than necessary, with expressions like $= (+(2, 2), 4)$.

Some alternative notations to Functional Form are discussed in appendix C.

Armed with all this terminology we are in a position to make proper (recursive) definitions of terms and propositions.

Definition 3.1 *Terms and Propositions*

1. *A variable or a constant is a term.*
2. *If $t_1, \ldots.., t_n$ are terms, f is an n-ary function and p is an n-ary predicate then*

   > $f(t_1, \ldots.., t_n)$ *is a term*
   > *and*
   > $p(t_1, \ldots.., t_n)$ *is a proposition*

3. *Only expressions formed using rules 1 and 2 above are terms or propositions.*

Our representation of formulae as expression trees can be extended to propositions and terms. For instance, the expression $\sin(90 - X) = \cos(X)$ can be represented by the tree in figure 3.1.

Here the tips are labelled by constants or variables and the other nodes by functions or predicates.

Where does all this leave our initial example proposition, 'n is prime'? Since this expresses a proposition which may either be true or false it must contain a predicate. Let us invent a new predicate, is_prime. Let it be a unary predicate, which takes in a number and returns true if and only if that number is prime. Then our proposition can be represented by

$$is\_prime(N)$$

where n is represented as the variable N. (Should n be a variable or a constant? Does 'n is prime' assert all numbers are prime or only one particular number?)

3.2 The Status of Variables

Variables in mathematical expressions often have an ambiguous status, whose resolution depends on the context. For instance, the X in

$$(X+Y)\cdot(X-Y)=X^2-Y^2 \tag{ii}$$

is usually intended to stand for any X and similarly for Y. We will call this the *universal* usage. However, the X in

$$\text{Solve} \quad X^2+2\cdot X+1=0 \ \text{ for } X \tag{iii}$$

stands for some particular number, whose exact value is not yet revealed. We will call this the *existential* usage.

Mathematicians sometimes try to resolve this ambiguity by calling the first equation an *identity* and using the predicate $\equiv$ instead of $=$. But this will not do because there are equations in which different variables are to be interpreted in different ways. Consider, for instance,

$$\text{Solve} \quad A\cdot X^2+B\cdot X+C=0 \ \text{ for } X \tag{iv}$$

Here the A, B and C stand for any number, whereas the X stands for some particular number, whose precise value will depend on A, B and C.

We will resolve this ambiguity by announcing the status of each variable with the aid of a *quantifier*, either a *universal quantifier* or an *existential quantifier*. A universal quantifier consists of an upside-down A followed by the variable whose status is being announced, e.g. $\forall X$. An existential quantifier consists of a back-to-front E followed by the variable, e.g. $\exists X$. Thus if we want to assert that (ii) holds for all X and Y we will write

$$\forall X \forall Y \ \ (X+Y)\cdot(X-Y)=X^2-Y^2$$

On the other hand, if we want to assert that (iii) has a solution we will write

$$\exists X \ \ X^2+2\cdot X+1=0$$

In the case of (iv) the precise value of X depends on A, B and C and this can be represented by the order of the quantifiers

$$\forall A \forall B \forall C \exists X \ \ A\cdot X^2+B\cdot X+C=0$$

Putting the existential quantifier for X before the universal quantifiers would imply that all quadratic equations had the *same* solution.

Some of the most subtle questions of variable status occur in Analysis. Consider the assertion that $1/X$ tends to ∞ as X tends to 0. This is usually expressed in English as

> For all M, there exists a Δ such that $|X| \leq \Delta$ implies that $|1/X| > M$

Expressed in our notation this becomes

$$\forall M \exists \Delta \forall X \;\; |X| \leq \Delta \rightarrow |1/X| > M \tag{v}$$

Note how (as implied by the definition) Δ depends on M, but not on X

Exercise 6 *Consider the following assertions that e^x is continuous and uniformly continuous. Express each of them in our formal notation. Compare the resulting formulae.*

> *Continuous - At each point x, for any positive ϵ there exists a δ such that $|e^y - e^x| < \epsilon$ whenever $|y - x| < \delta$.*
>
> *Uniformly Continuous - For any positive ϵ there exists a δ such that $|e^y - e^x| < \epsilon$ whenever $|y - x| < \delta$.*

In section 2.2 we gave a definition of 'formula'. This must now be updated to include the quantifiers. The first step is to retract clause 3 of the definition, which restricted the formulae to those defined using the connectives - we are going to add a new way to form formulae, using quantifiers. For the sequel we will find it useful to pay attention to the variables whose status has not been announced with quantifiers; the *free variables*. This entails redoing the definitions of terms and propositions giving the set of free variables they contain. We use the standard curly bracket notation for sets, e.g. the set containing just X, Y and Z is written $\{X, Y, Z\}$.

Definition 3.2 *Terms, Propositions, Formulae and Sentences.*

1. *A constant is a term with free variables $\{\}$*
2. *A variable, X, is a term with free variables $\{X\}$.*
3. *If $\mathbf{t}$ is a vector of n terms with free variables* **vars**, *v is the union of these variables, f is an n-ary function and p is an n-ary predicate then*

   $f(\mathbf{t})$ is a term with free variables v
   and
   $p(\mathbf{t})$ is a proposition with free variables v

4. *Only expressions formed using rules 1, 2 and 3 above are terms or propositions*
5. *A proposition with free variables v is a formula with free variables v.*
6. *If A and B are formulae with free variables va and vb then $\neg A$ is a formula with free variables va and $A \wedge B$, $A \vee B$, $A \rightarrow B$ and $A \leftrightarrow B$ are formulae with free variables $va \cup vb$.*
7. *If A is a formula with free variables v containing X then $\forall X A$ and $\exists X A$ are formulae with free variables v minus X.*

8. *Only expressions formed by rules 5, 6 and 7 are formulae.*

9. *A formula with no free variables is a sentence.*

The sentences, or formulae with no free variables, are of especial interest to us because they are the only formulae in which the statuses of the variables are unambiguous and thus are the only formulae which can be unambiguously assigned a truth value, t or f. Even though the status of a free variable is ambiguous, the universal status is the preferred reading. The sentence obtained from a formula by attaching a universal quantifier to it for each of its free variables is called its *closure*. In subsequent chapters we will often use an unquantified formula as an abbreviation for its closure.

3.3 The Meaning of Formulae

When is a sentence containing quantified variables true and when false? If symbols like $=$, 0, etc are interpreted in the conventional way then it is clear that

$\forall X \;\; X = X$ is true
$\forall X \;\; X = 0$ is false
$\exists X \;\; X = 0$ is true
and $\exists X \;\; X = X + 1$ is false

For instance, $\forall X \quad X = X$ is true because $0 = 0$, $1 = 1$, $2 = 2$, etc are true. $\exists X \quad X = 0$ is true because if X were 0 then $X = 0$ is true. A formula A', formed by substituting objects from a set, e.g. the natural numbers, for all free variables in A' is called an *instance* of A, e.g. $2 + 2 = 4$ and $2 + 2 = 5$ are both instances of $X + Y = Z$. Our intuitive interpretation of the meaning of the quantified formulae above used the rules,

- $\forall X \;\; A$ is true iff every instance of A is true.
- $\exists X \;\; A$ is true iff some instance of A is true.

From the s in section 2.1 we can deduce similar rules for the connectives, e.g.

$A \wedge B$ is true iff A is true and B is true.

Thus the truth value of a sentence can be recursively reduced to the truth value of each member of a collection (possibly infinite) of propositions not containing variables.

The meaning of formulae, which are not sentences, can be defined as follows. A formula will be said to be true iff all its instances are true (i.e. iff its closure is true). A formula will be said to be false iff all its instances are false . This leaves a lot of unsettled cases with some true instances and some false ones. In particular, note that a formula which is not true is not necessarily false and that a formula which is not false is not necessarily true.

We will find it useful to be able to manipulate substitutions as entities in their own right. A *substitution* is a set of pairs, where each pair is a variable, X, and the term, t, substituted for it. If we represent such a pair by t/X then the substitutions which form $2 + 2 = 4$ and $2 + 2 = 5$ from $X = Y$, can be denoted by $\{2+2/X, 4/Y\}$ and $\{2+2/X, 5/Y\}$. We can read t/X as 't is substituted for X' or 'X is *bound* to t'. We will let Greek letters like, ϕ , θ and ψ, stand for arbitrary substitutions. We will denote the result of applying a substitution, ϕ, to the formula, A, by writing the substitution immediately after the formula, i.e. $A\phi$. Using this notation

$$(X=Y)\{2+2/X, 4/Y\} \equiv (2+2=4)$$

where $\equiv$ means 'is identical to'.

Note that bound variables are immune from substitution. For instance, substituting 0 for X in $\forall X\ X = X$ causes no change.

$$(\forall X\ \ X=X)\{0/X\} \equiv (\forall X\ \ X=X)$$

Note also that instances are formed by substituting objects from some set (e.g. 1, 2, 3) for all free variables in a formula, rather than substituting arbitrary terms (e.g. $2+2$, $2+X$).

3.3.1 Interpretations

This discussion leaves an open question

> What objects must we consider substituting for the variable X in A when calculating the truth value of $\forall X\ \ A$ or $\exists X\ \ A$.

The truth value of the formula may hang on our choice. Consider the formula

$$\forall X\ \exists Y\ \ Y+1=X$$

This is true over the integers, but false over the natural numbers. That is, if we consider substituting integers for the variables, X and Y, then whatever integer is substituted for X, the number 1 less can be substituted for Y to make a true instance. However, if we are not allowed negative numbers then there is no number to substitute for Y to make $Y+1=0$ true. Similarly, the formula

$$\forall X\ \forall Y\ \ \{X < Y \rightarrow \exists Z\ \ (X < Z \wedge Z < Y)\}$$

is true over the reals, but false over the integers.

We need a relative notion of truth rather than an absolute one. In future we will talk about truth in an *interpretation*, where the interpretation is a system which provides

(a) A set of objects for the variables to range over. For instance, this might be the set of natural numbers, integers or reals, or it might be a finite set of objects. We will call it the *universe* of the interpretation.

(b) An assignment of meaning to the functions and predicates that appear in the formula, e.g. $+$, $=$, etc. We will want to be able to specify the conventional meanings we appealed to informally above, and some unconventional meanings.

Part (a) of the interpretation is provided by naming a non-empty set of objects as the universe, e.g. the natural numbers, $\{0, 1, 2, 3, 4, 5, \ldots\}$.

Part (b) of the interpretation is provided by associating *calculation procedures* with the constants, functions and predicates of the formula.

$+$	0	1	$=$	0	1
0	0	1	0	t	f
1	1	0	1	f	t

Figure 3.2: Calculation Procedures for $+$ and $=$

- The calculation procedure associated with a constant is trivial; it consists of assigning it a member of the universe as its *value*. We will often construct universes from the constants of a theory and assign a constant to itself as its own value.

- The calculation procedure associated with a function should specify the result of applying the function to all combinations of parameters drawn from the universe. This result will be another object in the universe.

- The calculation procedures associated with a predicate should specify the truth value obtained by applying the predicate to all combinations of parameters drawn from the universe.

One way of specifying these calculation procedures is to give tables like those in figure 3.2. Another way is to give a computer program.

Note that the tables defining each function and predicate give one and only one result for each combination of parameters from the universe. Thus this method of interpreting formula does not allow the possibility of multiply defined functions, e.g.

$$\sqrt{4} = 2 \quad \text{and} \quad \sqrt{4} = -2$$

or ambiguous predicates, nor does it allow the possibility of functions and predicates undefined in some region, e.g. $\frac{2}{0}$ must be assigned some default value. The assignment of a default value to normally undefined regions of functions and predicates can be something of a practical nuisance.

3.3.2 Interpreting Formulae

For instance, consider the formula

$$\forall X\, \exists Y \;\; X = Y \vee X = Y + 1$$

To keep matters simple we will use an interpretation with the finite universe, $\{0, 1\}$, and assign the constants 0 and 1 to themselves. The formula includes the binary function, $+$, so we must specify the results of applying $+$ to 0 and 1 in all possible ways, and these results must always be either 0 or 1. One way of doing this is given by the addition table in figure 3.2. The formula also includes the binary predicate, $=$, so we must specify the truth value obtained by applying $=$ to 0 and 1 in all possible ways. A way of doing this is also given in figure 3.2. We will call the interpretation so defined *boole*.

The meaning of our formula is determined by boole.

$\forall X \exists Y \;\; X = Y \vee X = Y + 1$ is true in boole
iff (by meaning of $\forall$)
$\exists Y \;\; 0 = Y \vee 0 = Y + 1$ is true in boole and

$\exists Y \;\; 1{=}Y \vee 1{=}Y{+}1$ is true in boole
  iff (by meaning of $\exists$)
either $0{=}0 \vee 0{=}0{+}1$ or $0{=}1 \vee 0{=}1{+}1$ is true in boole and
either $1{=}0 \vee 1{=}0{+}1$ or $1{=}1 \vee 1{=}1{+}1$ is true in boole
  iff (by semantic tree for $\vee$)
either $0{=}0$, $0{=}0{+}1$, $0{=}1$ or $0{=}1{+}1$ is true in boole and
either $1{=}0$, $1{=}0{+}1$, $1{=}1$ or $1{=}1{+}1$ is true in boole
  iff (by table for $+$)
either $0{=}0$, $0{=}1$, $0{=}1$ or $0{=}0$ is true in boole and
either $1{=}0$, $1{=}1$, $1{=}1$ or $1{=}0$ is true in boole
  iff (by table for $=$)
either *t*, *f*, *f* or *t* is true in boole and
either *f*, *t*, *t* or *f* is true in boole
  iff (by meaning of English)
t

When we design some axioms to capture a mathematical theory, for which there are established calculation procedures, say the theory of natural numbers, then we will want all the axioms to be true in the interpretation defined by these procedures. It will be convenient to be able to refer to this standard interpretation. We will call it *arith*. That is, arith is the interpretation with the natural numbers as universe and the standard arithmetic calculation procedures associated with $=$ (equality), . (multiplication) and $+$ (addition).

3.3.3 Some Definitions

When a formula is true in an interpretation the interpretation is said to be a *model* of the formula. This notion can be extended to a set of axioms by considering the formula made by conjoining all the axioms together (taking care not to get any accidental coincidences of variables). Thus a set of axioms can have a model. An axiomatization of the arithmetic of the natural numbers would have arith as a model.

Consider how each of the interpretations of a sentence might treat it. The following cases can arise.

1. If *all* the interpretations are models of the sentence then we say that the sentence is *logically valid*. Otherwise, the sentence is *logically invalid*.

2. If *none* of the interpretations are models of the sentence then we say the sentence is *unsatisfiable*. Otherwise, the sentence is *satisfiable*.

Note that a sentence can be satisfiable without being valid and invalid without being unsatisfiable. For example,

- $\forall X \;\; (X = 0 \rightarrow X = 0)$ is logically valid, because all its instances are tautologies.

- $\forall X \;\; X{=}0$ is logically invalid, because arith is not a model.

- $\forall X \;\; (X{=}0 \wedge \neg X{=}0)$ is unsatisfiable, because all its instances are contradictions.

- $\forall X \;\; X{=}X$ is satisfiable, since arith is a model.

The notion of a logically valid sentence extends the notion of a tautology, that we met in section 3. In fact, the logically valid sentences constitute the theorems of the mathematical theory, *Predicate Logic*, also known as *First Order Logic*. Similarly, the notion of an unsatisfiable sentence extends the notion of a contradiction. We establish this connection more formally in chapter 16.

3.4 Identifying Correct Arguments - Part 2

Can we now use the extended definition of the meaning of a formula, to deal with the remaining argument forms of section 1.3 – the non-boolean ones? Consider, for instance, the substitution rule:

$$\frac{A(X)}{A(T)} \tag{vi}$$

The first thing to note is that this does *not* mean the same thing as

$$A(X) \to A(T) \tag{vii}$$

In fact (vii), while plausible at first sight, is not logically valid: In fact it is not true even in the standard interpretation, arith. This may be clearer in a particular case. Let $A(X)$ be $X=0$ and T be 1 then (vii) is

$$X=0 \to 1=0$$

However, one of the instances of this is

$$0=0 \to 1 == 0$$

under the substitution $\{0/X\}$; and this is false in arith, since arith assigns $0=0$ to t and $1=0$ to f and $t \to f$ is f.

(vi), however, does not suffer this defect. It asserts that $1=0$ can be deduced from $X=0$, which is fair enough since $X=0$ means 'all numbers are equal to 0'. It does not allow us to deduce $1=0$ from $0=0$.

This means that we cannot represent arguments as implications between and and test for tautologyhood, as we did in section 2.2.4. Instead we must interprete the hypothesis and conclusion separately. We will use the following criterion

Definition 3.3 *An argument*

$$\frac{H}{C}$$

is correct iff every interpretation of H and C which is a model of H is also a model of C. We will say that C is a logical consequence of H.

If, in addition, H is a logical consequence of C, we will say that they are logically equivalent. This extends the definition of section 2.2.2 to formulae with variables.

Consider how we may use this definition to show the correctness of the substitution rule

$$\frac{A(X)}{A(T)}$$

There is a caveat to attach to this rule; T should not contain a free variable which is bound in $A(X)$. For instance, we are not allowed to substitute $Y+1$ for X in $\exists\ Y=X$, giving $\exists Y\ \ Y=Y+1$.

Let M be a typical model of $A(X)$ (which is also an interpretation of $A(T)$). We must show that M is also a model of $A(T)$, i.e. that $A(T)$ is true in M. Now every instance of $A(T)$ is also an instance of $A(X)$ and hence true in M. Therefore $A(T)$ is true in M. QED.

In a similar manner we can show the correctness of the other arguments from section 1.3.1. Consider, for instance,

This is an argument.
This is not propositional.
This is not syllogistic.

Therefore, some arguments are neither propositional nor syllogistic.

This can be formalized as

$$\frac{\text{argument(this)} \wedge \neg\ \text{propositional(this)} \wedge \neg\ \text{syllogistic(this)}}{\exists X\ \text{argument}(X) \wedge\neg\ [\text{propositional}(X) \vee \text{syllogistic}(X)]}$$

The universe of any interpretation of the hypothesis contains the object, 'this', hence the calculation procedures for the predicates must assign meanings to the propositions:

argument(this),
propositional(this) and
syllogistic(this)

Any model of the hypothesis must assign the first of these to *t* and the second and third to *f*. Thus, by the semantic trees for $\vee$, $\neg$ and $\wedge$, the instance of the conclusion defined by the substitution

–this/X˝

will be true and hence the conclusion will be true in all models of the hypothesis. QED.

Exercise 7 *Show the syllogism of section 1.3.1 is a correct argument form.*

Exercise 8 *Show that* $\exists X\ \ \neg A(X)$ *is a logical consequence of* $\neg\forall X\ \ A(X)$.

The technique illustrated above, for testing the correctness of non-boolean arguments, cannot be as easily translated into a computer program as the boolean technique of section 2.2.4 could. We cannot always test all models of a hypothesis because there may be infinitely many of them and testing the conclusion may involve considering an infinite number of instances. Thus the technique does not constitute a decision procedure, because it may not terminate; it may grind on forever without giving an answer. It can be shown that non-boolean arguments constitute an essentially *undecidable* area of Mathematics, i.e. there could not be a decision procedure.

The arguments above either deal with finite cases or exploit some special trick. The best we can do in general is to design a computer program which is guaranteed to terminate only if the argument is correct. This is done by trying to show that the theory consisting of the hypothesis and the negation of the conclusion is unsatisfiable, i.e. has no models. If the conclusion is not a logical consequence of the hypothesis then the model testing may end with a counterexample, but it may go on forever, inconclusively. Fortunately, if the conclusion *is* a logical consequence of the hypothesis this eventually becomes apparent. This is achieved by doing the model testing by a process of deducing theorems of the theory; the deduction of f demonstrates that the theory is unsatisfiable. Programs, which are guaranteed to terminate only if the argument is correct, are called *semi-decision procedures*. In chapter 5 we look at how theorems of a theory can be deduced, and in chapter 6 we investigate semi-decision procedures for logical consequence.

3.5 Summary

In this chapter we have looked inside the proposition and defined its internal structure. We found constants, variables, functions and predicates, and gave formal definitions of how these parts could be combined to form terms and propositions. We considered the status of variables, and showed how to distinguish the cases that arise by using quantifiers. This new notation was given meaning by introducing the idea of an interpretation.

We considered non-boolean arguments, which relied on the internal structure of the propositions. Such arguments were defined to be correct when the conclusion was a logical consequence of the hypothesis. This definition was used to show some arguments correct, but in general the procedure to do this is not a decision procedure; it is not guaranteed to terminate.

Chapter 4

Miscellaneous Topics

- This chapter consists of three loosely related sections.
- Section 4.1 is an introduction to higher order logics, especially typed lambda calculus.
- Section 4.2 gives the axioms for 3 mathematical theories: equality 4.2.1, group theory 4.2.2 and Peano arithmetic 4.2.3.
- Section 4.3 is a series of practical hints on how to represent knowledge using mathematical logic.

4.1 Higher Order Logics

The logical notation introduced so far is that of the of Frege, sometimes called . This will be enough for most of our purposes, but occasionally we will require something more.

4.1.1 Variable Functions and Predicates

Suppose we want to assert that if two objects, X and Y, are equal then they have the same properties. We will want to use a variable predicate, P, to stand for any property.

$$\forall P\, \forall X\, \forall Y \;\; X = Y \rightarrow \{P(X) \leftrightarrow P(Y)\}$$

Similarly, we might want to say that equal objects have equal values under the same function. We will want to use a variable function, F, to stand for any unary function.

$$\forall F\, \forall X\, \forall Y \;\; X = Y \rightarrow F(X) = F(Y)$$

First Order Logic only allows variables ranging over objects, e.g. X ranging over 90, e, etc. A logic in which variable functions and predicates are allowed and can be quantified over is called *Second Order*.

4.1.2 Functionals

Where there is a first and second, there must be a third. Consider the differential expression

$$\frac{d\ \cos(X)}{dX}$$

What sort of beast is the differential operator, $\frac{d}{dX}$? Clearly it is not a constant, variable or predicate. Is it a function? If it is, what are its parameters? Well it is dependent on the term $\cos(X)$, so this must be one parameter, and we must know what this term is to be differentiated with respect to, so X must be another parameter. Hence we could standardize the differential notation to:

$$d(\cos(X), X)$$

Unfortunately, this will not do. The X above is not really a free variable. For instance, we cannot substitute 90 for X and get a meaningful expression.

$$\#\frac{d\ \cos(90)}{d90}$$

(A # sign in front of an expression indicates that the expression is illegally formed.)

We may also change the variable of differentiation without changing the value of the expression.

$$\#\frac{d\ \cos(X)}{dX} = \frac{d\ \cos(Y)}{dY}$$

One way out of this difficulty is remove the variable of differentiation, and make $\frac{d}{dX}$ a function of a function, e.g. d takes cos to sin.

$$d(\cos) = \sin$$

Such a function is called a *functional*. Allowing *variable* functionals and quantification over them takes us into *Third Order Logic*.

Similar problems occur with integration and limits, and can be solved in similar ways.

Exercise 9 *What notation might we use to represent the limit of a function, $f(X)$, as X tends to some value l?*

Exercise 10 *Give a general definition of what it means for the limit of a unary function $f(X)$ to be ∞ as $x \to l$.*

4.1.3 Lambda Abstraction

Functionals apply to functions, rather than to terms, e.g. to sin rather than to $\sin(X)$. This is all very well when we have a convenient symbol to apply it to, but what does d apply to in the differential:

$$\frac{d\ \sin(X).e^X}{dX}? \tag{i}$$

We do not want to be constantly introducing new functions, e.g.

$$foo(X) = sin(X).e^X$$

just so we can replace (i) above by

$d(foo)$

A solution to this is the notion of *lambda abstraction*. We can turn an arbitrary term into the corresponding function by the following device.

$\lambda X(\sin(X).e^X)$ is foo

So now (i) above can be written as:

$d(\lambda X(\sin(X).e^X))$

4.1.4 Omega Order Logic

Now we can iterate the process, allowing functions of functionals (*Fourth Order*), functions of functions of functionals (*Fifth Order*) and so on. Allowing all of these then takes us to *Omega Order Logic*.

A nice way of capturing all the sorts of functions, functionals etc is provided in the *Typed Lambda Calculus* [17], which is one version of Omega Order Logic. Each of the constants is assigned a type. For instance, 90, e and $\frac{1}{2}$, may all be assigned type *real*, for real number. and t and f may be assigned the type *truth*, for truth value.

- Unary functions, e.g. sin, are then mappings from type real to real. If we use the arrow, $\mapsto$ to represent mappings, then this can be summarized as $real \mapsto real$.
- Binary functions, e.g. $+$, have type $real \times real \mapsto real$.
- The functional d has type $(real \mapsto real) \mapsto (real \mapsto real)$, that is, unary function to unary function.
- Binary predicates, e.g. $=$, have type $real \times real \mapsto truth$.
- Binary connectives (e.g. $\wedge$) have type $truth \times truth \mapsto truth$.
- $\lambda X \sin(X).e^X$ has type $real \mapsto real$, since X has type $real$ and $\sin(X).e^X$ has type $real$.

Thus all the various expressions we have met so far: formulae, propositions, terms, functions, predicates, connectives, variables and constants can be defined in a uniform manner using the terminology of the Lambda Calculus.

Definition 4.1 *The Expressions*

1. *If s is an expression of type τ containing free variables, $X_1, ..., X_n$, of types, $\tau_1, ..., \tau_n$, respectively then $\lambda X_1 ... \lambda X_n s$ is an expression of type $\tau_1 \times ... \times \tau_n \mapsto \tau$ whose free variables are those of s minus $\{X_1, ..., X_n\}$*
2. *If f is an expression of type $\tau_1 \times ... \times \tau_n \mapsto \tau$ and $s_1, ..., s_n$ are expressions of type $\tau_1, ..., \tau_n$, respectively, then $f(s_1, ..., s_n)$ is an expression of type τ whose free variables are the union of those of f and $s_1, ..., s_n$.*

3. *If p is an expression of type truth containing free variable X then $\forall X\ p$ and $\exists X\ p$ are expressions of type truth whose free variables are those of P minus $\{X\}$.*

4. *Expressions can only be formed from constants and variables and by application of rules 1, 2 and 3 above. A constant has no free variables. The free variables of a variable X are $\{X\}$.*

4.2 Mathematical Theories

Now that we have developed our notation we can achieve our original aim of representing mathematical knowledge in a form suitable for manipulation by a computer program. We will give some sample sets of axioms, so that later we may consider how theorems may be proved from them.

4.2.1 Equality

$=$ is a predicate which crops up throughout mathematics, not just in arithmetic and algebra, but in group theory, set theory etc. Thus the axioms which define $=$, form a subset of the axioms of all these theories.

Each of the axioms of equality is sufficiently famous to have earned itself a name. The first is *reflexivity* – an object is always equal to itself.

$$X = X$$

The second is *symmetry* – it does not matter how the parameters of $=$ are ordered.

$$X = Y \to Y = X$$

The third is *transitivity* – that equality is inherited.

$$X = Y \wedge Y = Z \to X = Z$$

The fourth and fifth are very similar and share the name *substitution axiom*. They assert that having equal parameters ensures equal results.

$$X_1 = Y_1 \wedge ... \wedge X_n = Y_n \to f(X_1, ..., X_n) = f(Y_1, ..., Y_n)$$
$$X_1 = Y_1 \wedge ... \wedge X_n = Y_n \wedge p(X_1, ..., X_n) \to p(Y_1, ...Y_n)$$

In fact, these are *axiom schemata* – we need a version for each n-ary function, f, and each n-ary predicate, p, in our theory. Hence we will need different versions of the fourth and fifth axioms in different theories.

These substitution *axioms* are not to be confused with the substitution *rule*, which allows the substitution of terms for variables.

Exercise 11 *How many of the equality axioms are true for the inequality predicates, $\geq$ and $>$, between real numbers? What additional axioms are needed for inequality over those needed for equality?*

Note that these axioms guarantee that, as mentioned in section 3.3.1, functions are defined everywhere and are not multiply defined. We will call these properties of functions *existence* and *uniqueness*, respectively. For instance, if f is a unary function, its existence property is guaranteed by the theorem

$$\exists Y\ f(X) = Y$$

which is easily proved from reflexivity (let Y be $f(X)$), and its uniqueness property is guaranteed by the theorem

$$f(X) = Y \wedge f(X) = Z \rightarrow Y = Z$$

which is easily proved from symmetry and transitivity.

4.2.2 Group Theory

The group axioms are another important constituent of mathematical theories. To start with we will need the equality axioms, reflexivity, symmetry, transitivity and some versions of substitution – just which versions we consider below.

In addition, we will need the axioms to define an identity element, e,

$X = e \circ X$
$X = X \circ e$
where $\circ$ is a binary function.

and some axioms to assert that every element, X, has an inverse, $i(X)$,

$X \circ i(X) = e$
$i(X) \circ X = e$

and the *associativity* axiom for $\circ$.

$$X \circ (Y \circ Z) = (X \circ Y) \circ Z$$

The functions which occur above are e, i and $\circ$, and the only predicate is $=$. Since e is nullary its substitution axiom is just an instance of the reflexivity axiom, and so can be omitted. The substitution axiom for $=$ is a simple consequence of transitivity, symmetry and reflexivity and can also be omitted. The remaining substitution axioms are:

$X_1 = Y_1 \rightarrow i(X_1) = i(Y_1)$ and
$X_1 = Y_1 \wedge X_2 = Y_2 \rightarrow X_1 \circ X_2 = Y_1 \circ Y_2$

4.2.3 Natural Number Arithmetic

The last in our series of classic sets of axioms are the *Peano axioms* for the Arithmetic of the natural numbers, 0, 1, 2, 3, ... This theory also requires the equality axioms.

The key observation behind the axiomatization is that all the *natural numbers* may be generated by a process of *succession*, starting at 0 and incrementing in 1s. The process of adding a 1 to a number is called forming its *successor* and is denoted by the *successor* function, s, i.e. $s(X) = X + 1$.

By using the successor function we can avoid having to have an infinite number of constants in our theory, i.e. 0, 1, 2, 3, ... We have one constant, 0, and represent other numbers by applying s to 0 repeatedly, e.g. 3 is $s(s(s(0)))$.

As with group theory we start with the equality axioms. The first two arithmetic axioms assert that the process of succession will keep generating new numbers: it will never loop back, either to 0

$\neg 0 = s(X)$

or to some other number

$s(X) = s(Y) \rightarrow X = Y$

The next pair of axioms provide a recursive definition of addition in terms of successor. Before giving it let us think about the different kinds of definition we can make in mathematics:

- The simplest kind are *explicit definitions.* The function (or predicate) to be defined appears once, dominating one side of the equation (or double implication), with distinct variables in each parameter position, e.g. $X^2 = X.X$
- A definition which is not explicit is an *implicit definition.* The function may appear on both sides of the equation, or may have non-variable parameters, or may appear in a non-dominating position, etc. Even so, the implicit definition may lay down enough conditions to determine uniquely the function, e.g. the definition of square root over the numbers by $(\sqrt{X})^2 = X \wedge \sqrt{X} \geq 0$.
- A *recursive definition* is implicit, but in a format which guarantees that the defined function is uniquely determined. It relies on the fact that the objects in the domain can be generated by a process like succession.

The Peano recursive definition of $+$ in terms of s is by two cases: the second argument of $+$ is either 0

$$X + 0 = X \tag{ii}$$

or is the successor of some number

$$X + s(Y) = s(X + Y) \tag{iii}$$

In the second case $X + s(Y)$ is defined in terms of $X + Y$. This is, of course, a circular definition, but not a vicious circle, since we can use it to calculate the sum of any two numbers, e.g.

$$\begin{aligned} s(s(0)) + s(s(0)) &= s(s(s(0)) + s(0)) && \text{by (iii)} \\ &= s(s(s(s(0)) + 0)) && \text{by (iii)} \\ &= s(s(s(s(0)))) && \text{by (ii)} \end{aligned}$$

Thus $+$ is uniquely determined by (ii) and (iii), even though it is not explicitly defined.

The third pair of axioms are a recursive definition of multiplication in terms of addition. Again we have the same two cases: the second parameter of $\cdot$ is either 0

$$X \cdot 0 = 0 \tag{iv}$$

or is the successor of some number

$$X \cdot s(Y) = X \cdot Y + X \tag{v}$$

Again $X \cdot s(Y)$ is defined in terms of $X \cdot Y$.

Exercise 12 *Calculate* $s(0) \cdot s(s(0))$ *using* (iv), (v), (ii) *and* (iii).

The format of the recursive definitions of $+$ and $\cdot$ was the same:

$$F(X, 0, \mathbf{Z}) = G(X, \mathbf{Z})$$
$$F(X, s(Y), \mathbf{Z}) = H(F(X, Y, \mathbf{Z}), X, Y, \mathbf{Z})$$
where $\mathbf{Z}$ is a vector of n additional parameters

(except that $\mathbf{Z}$ was empty in the definitions of $+$ and $\cdot$). Such a definition is called *primitive recursive* and is a very common type of recursive definition. More general types of recursive definition can be obtained by allowing, for instance, simultaneous recursion on two variables, X and Y, i.e. $F(s(X), s(Y), \mathbf{Z})$ is defined in terms of $F(X, Y, \mathbf{Z})$.

Exercise 13 *Give a primitive recursive definition of exponentiation,* X^Y*, (for natural numbers) in terms of multiplication.*

Some versions of natural number arithmetic allow the addition of any number of new axioms provided they conform to the format for explicit or recursive definitions of new functions or predicates, e.g. these theories would allow as new axioms the definition of exponentiation you just designed in the above exercise.

Lastly we need an axiom which says that all the numbers are generated by the succession process. The axiom of mathematical *induction* has essentially this message, because it says that a formula is valid if it is valid for all numbers defined by the succession process.

$$P(0, \mathbf{Z}) \wedge \forall X \;\; [P(X, \mathbf{Z}) \rightarrow P(s(X), \mathbf{Z})] \rightarrow \forall Y \;\; P(Y, \mathbf{Z})$$

Note that induction is a second order axiom since P is a variable predicate.

There is a dual relation between induction and recursion which we will make explicit in chapter 11.

That completes our examples of axiom sets for this chapter. We will be considering further axiom sets as they arise in the context of particular Mathematical Reasoning programs. We complete this chapter with some hints on how to build axiom sets.

4.3 Some Practical Hints

Those who intend to build computational models of mathematical (or any other kind of) reasoning are well advised to learn the art of representing knowledge in a logical calculus. One of the main aims of this book is to teach that art and that is why we have invested a disproportionate amount of effort (by the standards of conventional logic textbooks, e.g. [47]) in explaining logical notation. To conclude this teaching we would like to summarize a few practical hints and standard pitfalls.

4.3.1 Function or Predicate?

We have a relationship, r, between some objects, x and y, which we wish to represent. Should we represent it with a function and equality, i.e. $r(x) = y$ or with a predicate, i.e. $r(x, y)$?

This will depend on r. As noted in section 4.2.1, the equality axioms give function notation two properties that predicate notation lacks: the existence of a suitable y given x and the uniqueness of this y, up to equality.

Suppose we want to represent the relationship between a formula, fm, and its proof, prf. Now given fm, prf is neither guaranteed to exist (since fm may not be a theorem) nor is it unique if it does exist (since a theorem may have several proofs). So we cannot represent the relationship by an equation

$$\#proof\_of(fm) = prf$$

On the other hand, fm *is* uniquely determined by prf – a proof proves one and only one theorem. So the relationship can be represented by the equation

$$theorem\_of(prf) = fm$$

where $theorem\_of$ is a function which extracts the theorem proved by a proof, and returns some default value, say 'undef', for terms which are not proofs.

The relationship between a sentence, *sent*, and a model, m, however, cannot be represented by a function in either direction. A sentence can have several models or none. An interpretation can be a model of several sentences. So we must be contented with a relation, i.e.

$$is\_model(sent, m)$$

If we must represent an essentially non-function relationship, r, between x and y, with a function, we can always employ the following trick: a function, f_r, is defined from x to the set Y of all y for which $r(x, y)$.

$$f_r(x) = \{y : r(x, y)\}$$

If there are no ys, such that $r(x, y)$ for some x, then $f_r(x) = \{\}$. If there are several then $f_r(x)$ is a non-singleton set.

4.3.2 An Advantage of Avoiding Functions

Despite the advice in the above section there is a strong advantage to be gained if an entire mathematical theory – axioms and theorems – can be expressed without the use of proper functions, but with only predicates, constants, variables and universal quantifiers. In section 16.2, we will see that arguments in such theories are essentially boolean and can be settled by testing a finite number of variable free, quantifier free formulae for tautologyhood using the techniques of propositional logic.

As an illustration, consider the following alternative axioms for group theory. These are modified from the axioms in section 4.2.2 by exchanging expressions of the form, $S \circ T = U$ and $i(S) = T$, for, $\circ(S, T, U)$ and $i(S, T)$, respectively.

$$\begin{array}{ll}
identity & \circ(e, X, X) \\
 & \circ(X, e, X) \\
 & \\
inverse & i(X, Y) \rightarrow \circ(X, Y, e) \wedge \circ(Y, X, e) \\
 & \\
associativity & \circ(X, Y, U) \wedge \circ(U, Z, W) \wedge \circ(Y, Z, V) \\
 & \quad \rightarrow \circ(X, V, W) \\
 & \circ(Y, Z, V) \wedge \circ(X, V, W) \wedge \circ(X, Y, U) \\
 & \quad \rightarrow \circ(U, Z, W)
\end{array}$$

Axioms like these have been used with great success, by George Robinson and Lawrence Wos, as a basis for an automatic theorem prover for group theory, [66]. This success must be attributable, at least in part, to the essential finiteness of the processes involved.

Naturally, there is a price to be paid for such success. The price is that not all theorems of group theory can be proved from these axioms. In particular, it is not possible to prove any theorem which expresses the existence or uniqueness properties of $\circ$ or i, e.g.

$$\forall X\ \exists Y\ \ i(X, Y)$$

For a way round this difficulty see chapters 10 and 14.

4.3.3 Variadic Functions and Predicates

There are some functions and predicates which we tend to think of as being able to take any number of parameters – of being of variable arity or *variadic*. This is sometimes expressed by writing a chain of parameters and functions (or predicates), e.g.

$$2 + 3 + 2$$
$$3 \cdot a \cdot \sin(x)$$
$$2 \cdot 3 = 3 \cdot 2 = 6$$
$$2 \leq a \leq 5$$

(Note that for this chain notation to be used unambiguously on functions they must be associative. Division, X/Y, is not associative, so $2/3/2$ is ambiguous between $(2/3)/2$ and $2/(3/2)$.)

Unfortunately, Predicate Logic does not allow the variadic functions; the arity of a function must be fixed, and we have fixed the arity of $+$, $\cdot$, $=$ and $\leq$ at 2.

We can regain something of the flexibility of variadic functions and predicates by the following device. The function or predicate in question is made unary but given a composite parameter; one containing all the original parameters. A *set* of the original parameters is the sort of composite parameter I have in mind, but this is not quite right. Two of the properties of a set are wrong: the elements of a set are unordered and multiple occurrences of an element are merged. The sort of composite parameter we want is a tuple, which is usually called a *list* in computer circles. In a list the elements are ordered and multiple occurrences of the same element are allowed. Lists are delimited with angle brackets and the elements are separated by commas, e.g. $<2, 3, 2>$. Our unary (variadic) $+$, is written as

$$+(<2, 3, 2>)$$

our unary (variadic) $=$, as

$$= (<2 \cdot 3, 3 \cdot 2, 6>)$$

and our unary (variadic) $=$, as

$$\leq (<2, a, 5>)$$

Although the order of the parameters of functions and predicates is important in general, it is actually immaterial for two of the examples above, $+$ and $=$, because they are commutative (or symmetric). Thus, it would actually be more convenient to use a composite parameter for such functions which built in commutativity, i.e. in which order was ignored, but multiple occurrences of elements were retained. Such a composite object has been invented in Artificial Intelligence: it is called a *bag*. (If you put two cans of baked beans and a pound of sausages in a shopping bag, they may get jumbled about, but no cans will disappear.) We will delimit bags by square brackets and separate the elements by commas, e.g. $[2, 3, 2]$. Our unary (variadic) $+$, is written as

$$+([2, 3, 2])$$

4.3.4 Representing Negation

Suppose we want to say that a sentence, sent, is unsatisfiable – has no model. A common error is to formalize this as

$$\#is\_model(sent, none) \tag{vi}$$

where '*none*' is a special object denoting non-existence. But (vi) means that sent *has* a model called *none*!

Suppose we had a theorem that f (false) was not deducible from a set of axioms with a model, expressed as:

$$is\_model(Ax, M) \rightarrow \neg deducible(Ax, f)$$

Then substituting *sent* for Ax and *none* for M and applying modus ponens we could deduce:

$$\neg deducible(sent, f)$$

which would not be an intended consequence of (vi).

The correct way to represent the information that sent has no models is:

$$\neg\exists M \;\; is\_model(sent, M)$$

or equivalently

$$\forall M \;\; \neg is\_model(sent, M)$$

4.3.5 The Importance of a Semantics

We have expended a lot of effort in this chapter on defining a *semantics* for all our logical expressions. That is, we have given recursive definitions of what sort of expressions are allowed and, with the aid of semantic trees, explained what they mean. This effort was not wasted. It was this we were relying on in the last section when we rejected a way of representing negation, which looked plausible, but could have stored up trouble for the future. It is this we will be using in the next chapter, when we *prove* that the theorem proving processes we design will work correctly.

This ability, to judge representations and processes, is not an idle luxury. The Artificial Intelligence literature abounds with plausible looking formalisms, without a proper semantics. As soon as you depart from the toy examples illustrated in the paper, it becomes impossible to decide how to represent information in the formalism or whether the processes described are reasonable or what these processes are actually doing.

Consider, for example, the *beta structures* of the Merlin system [48]. A beta structure is a way of describing concepts, e.g. in [21] the concept '*monkey*' is described by:

$$monkey: \quad (mammal\ (hands\ 4)\ tail\ intelligent)$$

which we are told 'means', $mammal$ further specified by $(hands\ 4)$, $tail$ and $intelligent$. $mammal$ here looks like the 'type' of any $monkey$. $hands$ looks like a function, which applied to a particular $monkey$ will always return 4, e.g. $hands(jacko) = 4$. $tail$ and $intelligent$, on the other hand, look like unary predicates, e.g. $tail(jacko) \wedge intelligent(jacko)$, meaning 'has a tail' and 'is intelligent', respectively.

Also in [21] ape is described by

$$ape: \quad (mammal\ (tail\ none)\ intelligent\ (hands\ 4)).$$

Here, however, $tail$ has started to behave like a function, e.g. $tail(washoe) = none$, where $none$ is an alias of 0.

A process of 'matching' is described by which new descriptions can be generated from old ones. Matching can be achieved in several ways, in particular, if the beta structures, $(heada\ furthera)$ and $(headb\ furtherb)$, describing a and b have the same $head$ word, i.e. $heada \equiv headb$, and the further specification of b is an extension of the further specification of a then b can be described by $(a\ extras)$ where $extras$ are the further specifications in $furtherb$ which are not in $furthera$.

This process applies to $monkey$ and ape above, since they both have the same head word, $mammal$ and $((tail\ none)\ intelligent\ (hands\ 4))$ is an extension of $((hands\ 4)\ tail\ intelligent)$. The extra bit is $(tail\ none)$, so ape can be described by

$$\#ape: \quad (monkey\ (tail\ none)) \tag{vii}$$

It is not clear whether (vii) is an intended consequence of matching. Its obvious interpretation, that apes are a type of $monkey$ without $tail$s, is strictly false in the real world, whereas the obvious interpretations of the original descriptions are not. However, it has a metaphorical interpretation that apes are rather like $monkey$s without $tail$s, which is reasonable.

The fault of the Merlin system is that it is not clear what the intended interpretation of the formalisms are, and while we may be able to guess one when the concepts described are everyday objects, we may lose this ability as things get more complicated. Worst of all we cannot judge the correctness of processes like matching, because the meanings of the structures they manipulate are unclear. As long as the meaning is left unspecified by the authors of Merlin a user is free to choose any meaning whatever for his descriptions: and this may prove fatal if his choice violates some implicit assumptions behind the matching process, e.g. the kind of 'mistake' exhibited above may be much more serious for him.

There are times when the meaning of even the toy examples given in [21] elude me. I could understand psum: $(+\ 3\ para1\ 4)$

as a term with dominating function $+$, where psum is presumably an acronym of 'partial sum'. However, one of the other Merlin processes returned the 'value' of this as

$(7\ para1)$

and here I am at a loss. What is 7? If it is a constant then how can it be further specified? Can it be a function or a type?

Note that these last two beta structures overthrow all our assumptions about what kind of thing a beta structure can be and make us look again at the matching process above. Is it still correct? Can it withstand another assault like this? Only a semantics could provide an answer.

4.4 Summary

This chapter has covered a variety of topics.

- We considered higher order logic, involving variable functions, predicates, functionals, etc. We looked particularly at the typed lambda calculus.
- We gave sets of axioms for: equality, groups and natural numbers.
- We gave some hints on representing mathematical knowledge, including: whether to use functions or predicates; how to represent variadic functions and predicates; how not to represent negation and why semantics is important.

Further Reading Suggestion for Part I

We have now introduced all the mathematical logic necessary to have a formal notation for describing mathematical knowledge to a computer. For further reading in mathematical logic the reader is referred to [47].