CHAPTER ONE

LOGIC: PROOFS AND MODELS

The main aim of this book is to show how artificial intelligence (AI) can be enriched by incorporating recent advances in formal logic. Most of what we shall be looking at concerns extensions of classical logic to cover topics such as reasoning about time, reasoning about knowledge, or reasoning with uncertain knowledge. We do not intend to write a general introduction to logic, since there are already plenty of good introductory logic texts. On the other hand, we do need to make the nuances of our particular ways of thinking about classical theory clear, and to present some important meta-results in order to be able to compare it properly with the extensions and alternatives we will be considering in the remainder of the book. We therefore start with a discussion of classical logic.

1.1 Representation plus entailment

There are a number of ways of characterising what logic is about - the study of valid arguments, the study of consistent sets of statements, the foundations of mathematics, and so on. Different people will take different views as to which of these is the best way of thinking about the subject, and some people may even reject some of them. As far as AI researchers are concerned, it is perhaps most appropriate to think of it as being about knowledge representation languages in which the notion of entailment can be captured. In other words, it is about the study of languages in which we can state facts and rules, and for which we have a means of determining what other facts follow from the ones we have stated. The main thing that distinguishes logical languages from other representation languages is that the rules of entailment and the rules governing the interpretation of statements are presented with great precision. Sufficient precision, in fact, to

enable us to derive rigorous proofs about properties of the language itself. We take this precision to be the defining characteristic that makes a language a *logic*, and we will be concentrating in this book on just such languages. In chapters 2 and 3 we will look in detail at two languages studied within classical logic, and we will show that these languages possess a number of desirable properties, but that both are inadequate or disappointing in some way or other. This should both give the reader a feel for the way such proofs about a language work, and also provide a reference point for considering the advantages and disadvantages of the alternative theories we look at later on.

In order to study a language, we have to be able to recognise whether or not a string of symbols is in fact an acceptable sentence in it; and for strings of symbols which are sentences, we have to know how to work out what they mean. For languages which are to be used for knowledge representation, it is also useful for us to be able to find out whether some sentence follows from some other set of sentences, so that we can perform inferences. All the languages we will be looking at are specified by three sets of rules. We need *syntactic rules*, to tell us what sequences of symbols are meaningful expressions of the language; *semantic rules*, to tell us how to interpret such meaningful expressions; and *inference rules*, to tell us about the relations between them. The syntactic rules are generally very uninteresting. We need to have them, since the meanings of expressions are specified systematically in terms of their syntactic structure, and the inference rules also apply to expressions by virtue of their syntactic structure. Furthermore, proofs about what can and cannot be expressed, or about what can and cannot be proved, are nearly all couched in terms of the various ways expressions can be built up from smaller expressions, so that the syntactic rules are particularly important when we come to try to prove things about languages. They are, then, important but not interesting (there may be an analogy here with work on computer processing of natural language, which sometimes seems to concentrate on syntactic processing to the exclusion of semantics. The point is that although it may be the semantics we are actually interested in, the only way we can get at it is via the syntax). The semantics of logical languages are generally given in terms of a *model*. We will see an exception to this when we come to consider Lorentzen's (1959) *game theoretic semantics* in Chapter 9, but apart from this one exception we will always provide the semantics via a model. The particular details of what a model is will vary from language to language, but the general notion is fairly simple. A model is no more than a collection of objects which have relationships to one another. The objects may be almost any kind of entity you care to mention - things in the real world, mathematical abstractions, ideas in people's minds, even the symbols of the language itself.

The relations between them may also be of a wide variety of types. The only requirement is that we should be able to systematically relate expressions of the language to entities and relations in the model.

Once we have the notion of a model, we can distinguish between two sorts of truth in a language. There is a weak version of truth, which we might call *truth in a model*. We will see that sentences of the second language we will be considering, called FOPC for *first-order predicate calculus*, are basically built up out of names for individuals, names for variables, names for relations, and logical operators. Models of FOPC consist of sets of individuals and relations between them. To a first approximation, a sentence of FOPC is true in a model just when the relations it names hold between the individuals it names. There is quite a lot of extra detailed wrapping to be put round this notion, but the basic idea is as we see it here. As languages get more complex, the structure of a model will get more complex, but truth in a model will essentially remain the same, namely that the relations referred to hold between the individuals referred to.

The simple notion of truth in a model is useful, but it is not really what we need. Logic is generally used in situations where we know a certain amount about how the world is, including various general rules, and we need to find out whether various statements about which we do not have direct information are also true. To take a trivial example, we might know that all cats get sleepy when it's very hot, that Cherry is a cat and that it is very hot, and we might want to know whether Cherry is sleepy. We have no direct information about the statement we are interested in, but it clearly follows from the facts and rules that we do have direct access to. The way we arrive at this conclusion is by realising that in all models in which the first three statements are true, the fourth one must also be. The notion of a statement being true in all models, or in all models in which some other set of statements are also true, is of central interest in logic. It is the way in which we validate our inference rules, and hence the basis of all inferences of new knowledge from old. We will generally reserve the term true for truth in a model, and call sentences which are true in all models (in which some other set of sentences are true) *valid* (with respect to the given set of sentences). Validity will also become more complex in detail as we move to more complex languages, but the general idea will remain the same.

As well as semantic rules, we have rules of inference. The rules of inference are formal patterns which allow us to infer sentences from other sentences. They are themselves derived from our intuitions about what the symbols of a given language are supposed to mean, and from the concrete way in which these intuitions are expressed in the semantic rules. However, when they are employed in the course of a proof, the reasons why they are accepted as rules of inference,

and the meanings of the expressions they are applied to, are completely irrelevant. Rules of inference are entirely formal rules which say that if you have a collection of expressions which fit a specified pattern then you can add some new expression to them.

It is common to refer to the semantic rules of a language as its *model theory* and the inference rules as its *proof theory*. It is clearly important to ensure that the proof and model theories coincide as closely as possible. Ideally, everything which was valid according to the semantics of the language would be provable via the inference rules, and everything which was provable would be valid according to the semantics. We say that a language is *complete* if its inference rules are powerful enough to enable us to prove everything which is valid according to its semantics, and that it is *sound* if everything which is provable via the inference rules is valid. All the languages we will look at in this book are sound. A language whose rules of inference permitted you to draw conclusions which were not valid would be virtually useless, since you would then have no reason to trust any conclusions which you arrived at using those rules, and hence there would be very little point in using them. It is less important for languages to be complete. It's reassuring if they are, since it gives you confidence that you really have got the right set of inference rules, but the fact that you know that there may be things that are valid but which you cannot prove should not make it impossible for you to rely on the results of any inferences which you do manage to make.

We will return to the precise details of the relations between validity and provability for two important languages in Chapters 2 and 3, where we will prove a number of *meta-theorems*, i.e theorems which tell us about properties of languages. Before we do that, we will see how they illuminate the difference between two of the traditional ways of viewing logic.

1.2 Alternative views

Books on logic are more or less forced to start by trying to give a definition of what logic actually is. This is rather a funny situation. Most books start by giving an overview of how the subject matter being considered is going to be attacked, of what particular topics are going to be dealt with, and so on, but it is not customary to feel compelled to explain what the subject itself is actually about. The reason why logic is an exception is that there are at least two widely held, and differing, views on what it is really about. These views eventually converge, but they provide different starting points and different interpretations of the significance of the results on which the convergence occurs. We can express them

informally as follows. On one view, exemplified in Hodges' (1969) introduction, logic is about the study of consistent and inconsistent sets of beliefs. It is about the conditions under which it is reasonable to hold a collection of beliefs simultaneously, and about when belief in some set of sentences seems to entail belief in some other sentence. On this view, the central focus of the subject is on the reasons for supposing that it is unacceptable for anyone to believe all of *cats get sleepy when it's hot, Cherry's a cat, it is hot* and *Cherry is not sleepy* at the same time, or that anyone who believed the first three of these ought to feel compelled to believe *Cherry is sleepy*. This view of logic, namely that it is about the consistency or otherwise of sets of sentences, puts the semantic rules first. It says that the inference rules are interesting only insofar as they provide a way of calculating which sentences are consistent with one another. The other view, which can be clearly seen in Robbin (1969) and Kleene (1967), emphasises the function of logic as the support for argumentation. The notion of a proof is central, and most of the results which are considered to be important concern the range of things for which proofs are available. The semantics from this point of view is a reference point for investigating properties of proofs - can all valid sentences be proved, can any invalid ones, and so on. Both approaches explore the relations between the proof and model theories, so it is hardly surprising that the results they come up with are the same. The difference is between those people who want to show that the proof theory conforms to the model theory and those who want to show that the model theory supports the proof theory. It clearly ought to make no serious difference to the results they come up with, but it may lead to different emphases, and it may lead them to provide slightly different versions of the language. We will remark on the differences in the ways a language may be presented after we have actually seen some details.

CHAPTER TWO

PROPOSITIONAL CALCULUS

The first language we will look at in detail is the *propositional calculus*. This is not a particularly useful language, since it lacks expressive power, but it has the advantage of being extremely simple. It therefore serves well as a vehicle for introducing the sorts of proof that are required when we want to prove meta-theorems about more realistic languages. The style of proof is similar to what we will need later, but the content is much less complex. The hope is that it will serve well as a model for the later proofs, not that we will be able to use it as a knowledge representation language for AI.

The propositional calculus is, as its name suggests, about propositions and relations between propositions. It is a formalisation of the sort of reasoning that is needed for inferring from the rule *if it is raining then people carry their umbrellas raised* and the fact *people are not carrying their umbrellas raised* the conclusion *it is not raining*. The main restriction on the propositional calculus is that there is no way of substituting particular entities which are known to satisfy some property into rules which refer to all items that satisfy it. In more concrete terms, it is not possible to express within the propositional calculus the chain of inference that leads from *if it is raining then everyone will have their umbrella up, John is a person, John does not have his umbrella up* to *it is not raining* because there is no way of substituting the name of the individual *John* into the general rule about *everyone*. This limitation does restrict the expressiveness of the propositional calculus, but it does not render it entirely useless.

We will consider three versions of propositional calculus, reflecting variations on the two general approaches to logic discussed above. The first version, known as a *Hilbert system*, is appropriate for the view that logic is about proofs. The second and third versions, a *Gentzen system* and a *tableau (or Beth) system* respectively, are appropriate for the view that it is about consistency and

entailment.

2.1 Propositional calculus (Hilbert system)

2.1.1 *Propositional calculus (Hilbert system): vocabulary and syntax*

We will call our first version of the propositional calculus $Prop_{hilbert}$. The syntax of $Prop_{hilbert}$ is extremely simple. It consists of a set of names for propositions, a single logical operator or connective, and a single rule for combining simple expressions via the logical operator to get more complex ones, using brackets to indicate the way expressions have been built up where there is any possibility of ambiguity. All the languages we will be looking at use round brackets (and) and square brackets [and] wherever necessary to structure expressions correctly. No significance attaches to the choice of whether to use round or square ones, it is just a matter of readability. We will generally use curly brackets { and } to denote sets of objects, either enumerating them as in $\{x_1, x_2, x_3\}$ or defining them by a property they are expected to have as in $\{x: x > 4\}$ (to be read as "the set of x such that x is greater than 4"). If we need brackets for anything else we will generally use angle brackets < and >. We will make no further mention of brackets unless we are going to use them for some special purpose - in particular it is to be assumed from now on that the standard round and square brackets can be used wherever necessary for clarity, but that they will always have to be matched up correctly (an opening square bracket cannot be paired with a closing round one, and vice versa). We will frequently call the expressions of $Prop_{hilbert}$ (and other languages when we come to consider them) *formulae*.

$Prop_{hilbert}$ is to be a Hilbert system, i.e. a system which is tailored for talking about what can and cannot be proved within the language, rather than for actually saying things and exploring entailments. To this end we want the language to be as simple as possible - we want to have as few logical connectives and as few inference rules as we can get away with. It turns out that all we need are the following:

(i) an infinite set of names for propositions. We choose to use p, q, and r, plus the infinite collection p_1, p_2, ...

These basic proposition letters are intended to be interpreted as the names for concrete propositions such as *it is raining, people have their umbrellas up* or *Cherry is asleep*. The inferences which we want to capture are ones that depend simply upon the way these propositions are connected together, rather than on any relationships between the concepts

they refer to.

(ii) a name, †, for a distinguished proposition which is known to be false. You can think of this as some proposition which you know is false, such as 1=2, or you can simply read it as "the false".

(iii) the connective →, which is intended to be read as "implies", or "if ... then ...". → can be used to combine simple proposition letters, as in p → q, or complex formulae, as in (p → q) → (p → (r → q)). In any formula of the form A → B, A is referred to as the *antecedent* and B as the *consequent*.

(iv) a single rule for combining expressions together. If A and B are arbitrary expressions, then so is A → B (with brackets included if necessary to maintain clarity).

We use A, B, C and A_1, A_2, ... to stand for arbitrary expressions of the language. It is important to recognise the difference between p, q, r and p_i, which are particular propositions of the language, and A, B, C and A_i, which are *meta-variables* which we use for talking about general expressions. A, B, C, A_1, ... are not part of the vocabulary of $Prop_{hilbert}$. They are a notation we use for talking about general properties of expressions which do belong to $Prop_{hilbert}$. Thus if we say "If A and B are expressions", we mean that they might be replaced by any of the particular proposition letters or by complex formulae built up from them by rule (iv). We sometimes use A, B ... to write *expression schemas*, i.e. patterns which a variety of actual formulae would match, so that A → B is a schema which would match p → q, p → r, (p → q) → [(r → s) → †], and so on.

$Prop_{hilbert}$ is a very impoverished looking language. The surprising thing about it is that it turns out to be sufficient for expressing anything which can be said in any language whose semantics are based on the valuation approach given below. It is thus well designed for use by people who want to prove meta-theorems about particular formalisms. The other versions of propositional calculus that we consider in this chapter are more useful for if you want to state facts and rules, and to use these for solving specific problems by proving theorems within the formalism. $Prop_{hilbert}$ is not, in fact, quite the simplest language we could have chosen, but it is simple enough.

2.1.2 *Propositional calculus (Hilbert system): semantics*

As with every single language that we will look at in this book, the semantics for $Prop_{hilbert}$ is given by explaining the meaning of the basic formulae (in this case the proposition letters and †), and then by explaining how the meanings of complex expressions depend on the meanings of their components.

The semantics of $Prop_{hilbert}$ are given in terms of two objects called T and F. It is evident that we intend these to correspond to the intuitive notions of truth and falsity, but nothing whatsoever depends on this. They could be 1 and 0, or 2999 and the name Cherry, or one of them could be an orange and the other a banana, or even the letters "T" and "F". It makes no difference, just so long as they are two different objects. The semantics of the basic letters and † are given by providing a function V, called a *valuation*, whose domain is the set of letters plus † and whose range is the set {T, F} (in other words, it takes as argument either one of the basic letters or † and returns either T or F). There are clearly an infinite number of such functions. The only constraint is that V(†) must be F.

V defines the meaning of the basic expressions, i.e. the proposition letters and the symbol †. The next move is to show how we build up the meaning of compound expressions from the meanings of the basic ones that appear in them. We will want some way of denoting the meanings of expressions in general. It does not seem to be quite correct to denote the meaning of an expression like $p \rightarrow (p \rightarrow q)$ by $V(p \rightarrow (p \rightarrow q))$, since V is simply a function which maps basic proposition letters and † to {T, F}. We will generally use the notation $[\![EXPR]\!]_T$ to denote the meaning assigned to EXPR by the theory of meaning T. $[\![\ldots]\!]$ are *meta-symbols*, for making statements *about* expressions of a language, rather than symbols of any of the particular languages we will be looking at. Whenever possible we will omit the subscript indicating the particular theory of meaning.

For $Prop_{hilbert}$, the semantics of basic expressions is clear. The basic expressions of $Prop_{hilbert}$ are the proposition letters and the symbol †. If P is either of these, $[\![P]\!]_V$ is just V(P).

Once we have the semantics for the basic expressions, we can provide it for complex ones. There is only one possible way of building a complex expression in $Prop_{hilbert}$, namely by combining two simpler ones with →. We extend the semantics of $Prop_{hilbert}$ by saying that if A and B are expressions then $[\![A \rightarrow B]\!]_V$ is calculated as follows. If $[\![A]\!]_V$ is T and $[\![B]\!]_V$ is F then $[\![A \rightarrow B]\!]_V$ is F, otherwise it is T. This particular choice for the meaning of → may seem counter-intuitive. We are saying that A → B is true so long as B is never false when A is true. It is important to bear in mind that we are not attempting to characterise the idea of *causality* in our treatment of →. Causality is a complex notion which is beyond the scope of this book. For almost the whole of the

current book we simply take A → B to indicate that A *materially implies* B, i.e. that it is not the case that B is false and A is true. This is a useful notion, but it must not be confused with the suggestion that A *causes* B.

The notion of a valuation can be given a very concrete realisation in terms of a *truth table*. A truth table is an enumeration of the way the valuation of a complex expression varies with the valuations of the simpler expressions it is made up from. For $Prop_{hilbert}$, the only truth table we need to consider is the one for expressions made by combining simpler expressions via →:

Fig. 2.1 Basic truth table for $Prop_{hilbert}$

A	B	A → B
T	T	T
T	F	F
F	T	T
F	F	T

We can construct truth tables for complex expressions by systematically building the tables for their components and then referring to the basic table given in Fig. 2.1. The truth table for A → (B → A), for instance, is constructed by first building the truth table for B → A for all values of A and B, and then combining these values in accordance with Fig. 2.1.

Fig. 2.2 Truth table for A → (B → A)

A	B	B → A	A → (B → A)
T	T	T	T
T	F	T	T
F	T	F	T
F	F	T	T

The final column in Fig. 2.2 is built by considering the first and third columns as though they were the first two columns in Fig. 2.1.

There are three particularly interesting schemas for $Prop_{hilbert}$. We can use these to show that this system is as expressive as any possible system whose semantics can be given either via a 2-valued valuation function or via simple truth tables, so that $Prop_{hilbert}$ is all we really need to consider as far as propositional logic is concerned. The following truth tables show that $Prop_{hilbert}$ can capture the important notions of negation, conjunction and disjunction.