Classical Logic: a Crash Course for Beginners

Antony Galton
Department of Computer Science
University of Exeter
Exeter EX4 4PT

June 25, 1997

Introduction

No doubt every reader of this journal is aware that Computer Science is becoming infiltrated by a strange breed of people called Logicians, who try to convince computer people that their arcane symbolism and obscure terminology are just what is needed to solve the Software Crisis, the Hardware Crisis, and any other difficulties that the computer world finds itself facing. Unfortunately the symbolism and the jargon can be very off-putting to anyone who has not already become immersed in Formal Logic; I have often met people who work with computers and are aware of how important logic is claimed to be by its devotees, and who feel that they really ought one day to make an effort to penetrate its mysteries, but who have not known how to set about doing so. This article and its sequel ('Logic as a Formal Method') are intended as a fairly gentle initiation into what Logic is about and what it has to offer Computer Scientists. They are inevitably very sketchy and incomplete—more like the brochures that can be picked up at a travel agent's than a proper guide-book—but it is to be hoped that some, at least, of my readers will come away with a clearer picture of what lies in store for them if they decide to follow up the more detailed references.

The present article outlines the two systems which form the standard core of formal logic, the Propositional Calculus and the Predicate Calculus. For a more detailed treatment, see my book *Logic for Information Technology* (Wiley, 1990). The second article looks at applications of these systems to Computer Science, and then expands the horizons by looking at some of the non-standard logical systems that have developed from the standard core and which have found application in a computational context.

1 The Propositional Calculus

Imagine a group of five people discussing which of them will go to town that afternoon. Only Anne and Barbara can drive, so one of them must go; Anne won't go without one of her sons, Charles and David, to keep her company. Charles would insist on taking his friends Elizabeth and Fiona along; on the other hand, if Fiona goes without David, then Elizabeth will insist on staying to keep him company. It then transpires that David cannot go, as he has too much homework to do. It follows that Barbara will have to go.

One does not need formal logic to reach this conclusion; but on the other hand it pays to be systematic, and formalism is a good way of achieving this in many cases. We express the data in terms of a set of primitive (i.e. not further analysed) propositions $AnneGoes$, $BarbaraGoes$, and so on, and a few basic connectives **if**. . . **then**. . . , **and**, **or**, **not**:

> $AnneGoes$ **or** $BarbaraGoes$
> **if** $AnneGoes$ **then** ($CharlesGoes$ **or** $DavidGoes$)
> **if** $CharlesGoes$ **then** ($ElizabethGoes$ **and** $FionaGoes$)
> **if** ($FionaGoes$ **and not** $DavidGoes$) **then not** $ElizabethGoes$
> **not** $DavidGoes$

with the desired conclusion

> $BarbaraGoes.$

For conciseness we write this as

$$\begin{array}{l} A \vee B \\ A \to (C \vee D) \\ C \to (E \wedge F) \\ (F \wedge \neg D) \to \neg E \\ \neg D \\ \hline B \end{array}$$

The system we are using is called the Propositional Calculus. The primitive elements of the system are ('schematic') letters standing for propositions (e.g., C for 'Charles goes') together with various connectives ($\vee$, $\wedge$, $\to$, and $\neg$) for linking them up to produce more complex propositions.

Formal Logic offers us several approaches to validating the inference above. In *Classical Logic*, the guiding principle is that the connectives in the inference are *truth-functional*. This means that, for instance, whether or not $A \wedge B$ is true depends only on whether or not A and B are: specifically, $A \wedge B$ is true provided that both A and B are true. Thus $\wedge$ corresponds to the use of the word 'and' in a sentence like 'Anne studies Physics and Barbara studies French', which is true just so long as both the sentences 'Anne studies Physics' and 'Barbara studies French' are true. Sometimes 'and' points to a more intimate connection between the sentences it joins: in 'Anne opened the door and Barbara entered the room' the word 'and' means something like 'and then as a result of that'. The logical symbol $\wedge$ cannot be used to record these additional components in the meaning of 'and'.

Similarly, $A \vee B$ is true just so long as at least one of A and B is true; and $\neg A$ is true so long as A is false. That leaves $A \to B$: although an expression of the form 'if A then B' normally suggests some sort of causal connection between the antecedent A and the consequent B, in classical logic we focus on the truth-functional core of its meaning, which is that you won't have the antecedent true as well as the consequent false: so we say that $A \to B$ is only false in the case that A is true and B is false. This means that it is true so long as either A is false or B is true. For example, the only situation in which the statement 'If you drop that glass it will break' is clearly false is one in which you drop the glass but it doesn't break—so in any other situation, i.e., in any situation in which either you don't drop the glass or it does break, the statement counts as true. (A fifth connective $\leftrightarrow$ is defined so that $A \leftrightarrow B$ is equivalent to $(A \to B) \wedge (B \to A)$; it is true just so long as A and B are either both true or both false.)

We now return to our inference, and consider just what we mean by saying that the conclusion follows from the premisses: it is that if the premisses are all true then the conclusion must be true too. So one way of checking the validity of the inference is to go through all the possible combinations of who goes and who doesn't, and check that in every case in which the premisses are true, the conclusion comes out true as well. To do this is to make a *truth table* for the inference. Since there are six different primitive propositions (A to F), and each of them can, in principle, be true or false independently of the others, this gives us $2^6 = 64$ different combinations to check.

Instead, we take a short cut. We are only interested in checking that the case of premisses true and conclusion false never occurs. So let's look for such a case. For the conclusion to be false we require that Barbara doesn't go: B must be false. For the first premiss to be true we need either A true or B true; so given that B is false that means we must have A true. The second premiss will be false if A is true and $C \vee D$ is false, so the latter proposition must be true too. That means that either C or D (or both) must be true. But D is false, from the fifth premiss, which means that C must be true. So by the third premiss E and F must both be true too, otherwise we would have C true and $E \wedge F$ false, which would make the third premiss false. We now have F true, D false, and E true: and this makes the fourth premiss false. What we have shown is that if the conclusion (B) is false then at least one of the premisses must be false as well. But this is just to say that if all the premisses are true then the conclusion must be true too: which is to say, the inference is valid.

We mentioned above that a systematic enumeration of all the possible combinations of truth and falsity of A, B, $\ldots$, F would enable us to validate the inference. What we did instead was to reason about what such a systematic enumeration would look like without actually doing it. All we need now is a systematic way of performing such reasoning. Any such systematic method is called a *Proof Theory* for the Propositional Calculus.

Proof Theories come in two flavours. One type, called *Derivation Systems*, enable one to validate an inference by deriving it from the premisses in a series of steps. The other type works more along the lines of our reasoning above: we start by assuming that the conclusion is false and see if we can make all the premisses true compatibly with this assumption. These are called *Refutation Systems* because they allow us to prove that a conclusion logically follows from some premisses by refuting (i.e., proving false) the assumption that we can have the conclusion false if the premisses are all true.

A proof theory is formalised by laying down in advance exactly what reasoning steps are allowed. This has to be done in terms of the *form* of the propositions we are manipulating, since while there is no limit to the possible subject-matter or *content* the propositions may have, we can specify exactly the range of possible forms—viz., every proposition (in the Propositional Calculus, of course) is either a primitive proposition or the negation of a proposition or the conjunction ($\wedge$), disjunction ($\vee$), conditional ($\rightarrow$), or biconditional ($\leftrightarrow$) of two propositions. The Propositional Calculus can only be used to validate those inferences which depend for their validity on the truth-functional structure of their component propositions.

The language of the Propositional Calculus can help us to be systematic when talking about computer programs. When a piece of program code is executed, the variables which occur in it are transformed from some initial set of values to a final set of values (when the execution is completed). To determine whether the program meets some previously given specification, we need to be able to state precisely both what it actually does and what it is supposed to do. Any system for doing this will benefit from the clarity provided by the Propositional Calculus; in addition, as explained below, the Propositional Calculus lies at the heart of all the formalised systems of reasoning procedures that we call logics, and it is to such systems that we must turn if we want to describe in a rigorous fashion what a program is intended to do, and to prove that it actually does it.

Consider, for example, the Pascal fragment

```
z:=1;
while x>0 do
      begin
      x:=x-1;
```

```
z:=z*y
end.
```

If executed with $x = 3, y = 4, z = 3$ initially, this program will terminate with $x = 0, y = 4, z = 64$, whereas if it is executed with $x = -4, y = 6, z = 3$ initially, it will terminate with $x = -4, y = 6, z = 1$. In general, if the initial values are $x = X, y = Y, z = Z$, then if $X \geq 0$ then the final values will be $x = 0, y = Y, z = Y^X$, whereas if $X < 0$ then the final values are $x = X, y = Y, z = 1$. Thus the program effects a transformation from a state in which the proposition

$$x = X \wedge y = Y \wedge z = Z$$

is true to a state in which the proposition

$$\begin{array}{l} (X \geq 0 \rightarrow (x = 0 \wedge y = Y \wedge z = Y^X)) \\ \wedge \\ (X < 0 \rightarrow (x = X \wedge y = Y \wedge z = 1)) \end{array}$$

is true. It is possible that this is exactly what the programmer intended, but the only way we can meaningfully decide this is if we have a statement of the programmer's intentions that is every bit as precise as the statement we have just given of the program's actual behaviour; for example, the intention might be to write a piece of code that transforms the initial state

$$x = X \wedge y = Y \wedge z = Z$$

to the state

$$(X \geq 0 \rightarrow x = X \wedge y = Y \wedge z = Y^X) \wedge (X < 0 \rightarrow x = X \wedge y = Y \wedge z = 0).$$

The discrepancy between the program and its specification is now clear.

2 The Predicate Calculus

The Propositional Calculus only gives us a very coarse-grained view of the logical structure of a statement or an inference. When we represent the statement 'If Fiona goes but David doesn't then Elizabeth won't go' by the schema $(F \wedge \neg D) \rightarrow \neg E$ we are only representing the structure that this statement has in common with a whole host of other statements such as

> If it's lunchtime but John isn't hungry then he won't eat anything.
> If you're a candidate in the election and nobody votes for you, you won't get elected.
> If all dogs eat meat and Fido doesn't eat meat then Fido isn't a dog.
> If X is prime and $X \neq 2$ then X is not even.

A more detailed account of the logical structure of these sentences is provided by the *Predicate Calculus*.

The Predicate Calculus is a variety of formal logic that can be used for formalizing discourse about domains consisting of discrete objects which may be described in terms of properties that are *strictly bivalent*, that is, a property must be defined so that each object in the domain either does or does not have it, with no "grey area" in between. It is important to note that everyday discourse is often not like this, and to the extent that it is not, the Predicate Calculus is an inappropriate tool for formalizing it. Sometimes, however, one can get acceptable results by *pretending* that an ill-defined property is in fact sharply defined, by stipulating an arbitrary "cut-off" point between objects which have the property and those which don't. For example, in the domain of human beings, the property of being *male* is strictly bivalent, in that every human being is either definitely male or definitely not male, whereas the property of being *tall* is not bivalent, since we cannot say definitely how high someone has to be to count as tall; but if we need to, we can arbitrarily lay down that anyone over six feet tall, say, is to count as tall. Similarly, in the domain of natural numbers, the properties of being *even*, or *prime*, or *greater than 100*, are all strictly bivalent, whereas the property of being *large* is vague in the same way as being tall is with human beings; but again, for some purposes it might be reasonable to define a large number as one which is greater than a million, say. Some other properties (on the domain of humans again), which are ill-defined in various degrees, are being a student, understanding logic, loving one's mother, having published an article in *Computer Journal*, being at school, and being awake.

In addition to *unary* properties such as maleness, tallness, primality, and so on, which apply to single objects, the Predicate Calculus also allows us to refer to *relational* properties, which hold between objects taken two at a time (*binary* relations), three at a time (*ternary* relations), or quite generally n at a time (*n-ary* relations) for any positive integer n. Examples of binary relations on the domain of human beings are: one person's being taller than, loving, speaking the same language as, or living next door to another person. On the domain of natural numbers we have such relations as one number's being greater than, being a multiple of, having no factors in common with, and being the square of another.

These examples show that we have a variety of different ways of expressing properties and relations in English. We can use nouns (as in 'being a *student*'), verbs ('*loving* one's mother), adjectives ('being *tall*') or prepositional phrases ('being *at school*). In the Predicate Calculus we circumvent this plethora of notational devices by using a uniform way of referring to properties. The standard way of ascribing a property to an object in the Predicate Calculus is to place a *predicate letter*, generally an upper-case letter such as P, Q, or R, to the left of a pair of parentheses, and in the parentheses to place an expression (called a *term*) denoting the object to which we wish to ascribe the property. For example, if we want to say that 7 is prime, we can stipulate that the predicate letter P is to denote the property of primality and that the term a is to denote the number 7, and write $P(a)$. And this same expression, with its constituents interpreted differently, will do equally as representations of the statements that Neil Kinnock is male, that Exeter is a cathedral city, and that 1992 is a leap year.

To express relational properties we use exactly the same method, except that if it is a binary relation we must put two terms, separated by a comma, in the brackets after the predicate letter—as for example $R(a, b)$, which might express that 4 is less than 9, that 9 is a factor of 45, that Terry Waite is taller than John Major, or that Cambridge is further east than Oxford—and in general for an n-ary relation we must put n terms in the brackets.

A *simple n-ary predicate* simply consists of a predicate letter together with n slots, or *argument-places*, into which we can insert terms to make a simple statement (or *atomic formula*). Thus the atomic formula $P(a)$ is the result of inserting the *constant term* a into the argument place of the predicate $P(\_)$. Once we have decided what property the predicate-letter P stands for, and what object a stands for, we then know how to assign a truth-value to the formula $P(a)$: it will be true just so long as the object denoted by a has the property denoted by P. In the so-called *formal semantics* of the Predicate Calculus, though, we don't say anything about properties, but content ourselves with the thought that any sharply defined property will divide the domain neatly into two mutually exclusive parts (or *subsets*), consisting of those objects which *have* the property in question, and those which do not. So instead of talking about properties, we talk about subsets of the domain, in effect replacing a property by the set of domain elements which have it. We can now say that the formula $P(a)$ is true just so long as the object denoted by a is a member of the *set* denoted by P.

We do a similar kind of thing with relational predicates. A binary predicate, for example, is intended to refer to a relation on the set of domain objects. It is a commonplace in mathematics that a binary relation can be represented as a set of ordered pairs, so that a pair (x, y) is included in the set just so long as x stands in that relation to y. So we can now say that once we have decided which objects a and b refer to, and which sets of ordered pairs R refers to, the atomic formula $R(a, b)$ will be true (under this interpretation) just so long as the ordered pair consisting of the objects denoted by a and b respectively is a member of the set denoted by R.

In addition to predicates, a first-order language may contain *function symbols*. These are expressions which enable us to create new terms out of old. In ordinary language we have expressions like 'the father of' which have just this role: for example, from the term 'John' we may, by successive applications of this function expression, construct the complex terms 'the father of John', 'the father of the father of John', 'the father of the father of the father of John', denoting the succession of John's male ancestors along the paternal line. Similarly, in mathematics we have notations like 'sin' which enables us to construct the term '$\sin \pi$' from the term 'π'. In addition we have *binary* functions such as '$+$' and '$/$', enabling us to construct terms such as '$34 + 56$' or '$\pi/3$'. In Predicate Calculus notation we notate functional expressions uniformly by the use of function symbols with argument places, exactly like predicates except that (a) we use lower-case letters instead of upper-case ones[1] and (b) the result of filling in the argument places of a function symbol with terms is not a formula but another term. Thus if we have a unary predicate letter P, a unary function symbol f, and a constant a, then $a, f(a), f(f(a)), f(f(f(a))), \ldots$ are all terms whereas $P(a), P(f(a)), P(f(f(a))), P(f(f(f(a)))), \ldots$ are all formulae (you can think of them as saying something like 'John is bald', 'John's father is bald', 'John's father's father is bald', and so on).

The formal semantics defines an *interpretion* of our formal language to consist of

- a specification of some domain, represented as a set of objects D,
- a specification of

    1. what element of D each constant term of the language is to stand for (the *denotation* of the term),
    2. what set of domain elements each unary predicate letter is to stand for, what set of pairs of domain elements each binary predicate letter is to stand for, and in general what set of n-tuples of domain elements each n-ary predicate letter is to stand for (the *extensions* of the predicates),
    3. what function on the domain each function symbol stands for (an n-ary function symbol will stand for a function from domain n-tuples to domain elements).

  This specification can be represented by an *interpretationfunction* I which maps each language element onto the domain-related entity that it is to stand for.

We can then say that an atomic formula $P(a_1, a_2, \ldots, a_n)$ is *satisfied* by the interpretation (D, I) just so long as the n-tuple $(I(a_1), I(a_2), \ldots, I(a_n))$ of domain elements denoted by the terms $a_1, a_2, \ldots, a_n$ is a member of the set of domain n-tuples $I(P)$ denoted by the predicate-letter P. If a formula ϕ is satisfied by an interpretation (D, I), we write

$$(D, I) \models \phi$$

[1] Warning: This is not an invariable convention.

and say that (D, I) is a *model* for ϕ.

All this may seem very trivial, and in a sense it is, but it does for the Predicate Calculus what the truth-tables do for the Propositional Calculus: it provides us with a crisp formal definition of the notion of an interpretation, and this in turn allows us to formulate precisely what we mean by general logical notions such as truth, consistency, and validity. Truth is represented by satisfaction: to say that a statement is true is to say that the formula representing it is satisfied by the interpretation that corresponds to the meaning of the statement; a set of statements is consistent so long as the set of formulae representing them is satisfiable, that is, are all satisfied together by some interpretation; and an inference is defined to be valid just so long as every interpretation which satisfies all the premisses also satisfies the conclusion.

In fact we have not so far revealed anything like the true power of the Predicate Calculus, which comes not from the construction of atomic formulae which we have outlined, but from the construction of complex formulae from these atomic ones. The Predicate Calculus furnishes us with two ways of constructing new formulae from old. One is just the same as in the Propositional Calculus, using the truth-functional connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$. The other, which is really the distinctive feature of the Predicate Calculus, makes use of two new symbols $\forall$ and $\exists$ known as *quantifiers*.

First, we must introduce the notion of a *complex predicate*. We have seen that a formula can be formed by inserting terms into the argument places of a predicate. But we can look at this the other way around too: we can form a predicate by taking a formula and *removing* one or more terms to leave argument places. For example, given the atomic formula $R(a, b)$, we can form two complex unary predicates $R(\_, b)$ and $R(a, \_)$. If we interpret $R(a, b)$ to mean that 8 is factor of 96, then the complex predicate $R(\_, b)$ will denote the property of being a factor of 96, while the complex predicate $R(a, \_)$ will denote the property of having 8 as a factor. We can form complex predicates out of non-atomic formulae too. For example, from the formula $P(a) \wedge Q(b)$ we can form the complex binary predicate $P(\_) \wedge Q(\ldots)$. But we can also stipulate that two argument places are to be filled by the same term; then the complex predicate $P(\_) \wedge Q(\_)$ is actually a *unary* predicate, since although it has two argument places, they are two copies of the *same* argument place and hence must be filled by the same term. So we can get $P(a) \wedge Q(a)$ from this predicate, but not $P(a) \wedge Q(b)$. If $P(\_)$ denotes the property of being even, and $Q(\_)$ denotes the property of being a square, then $P(\_) \wedge Q(\_)$ denotes the property of being an even square.

In general, a language does not have a name for each of the objects that it wants to talk about. We might want to talk about cows, but we do not have to have a name for every individual cow. We can still make statements which apply to every cow, for example, we can say ‘All cows eat grass’. If I say ‘Some cows have horns’, then here too I am implicitly referring to every cow, since I’m *denying* that every cow is hornless. In mathematics, when we talk about the real numbers we are talking about an indenumerable domain, and it is in principle impossible for a finitely-definable language to have indenumerably many names. So most real numbers cannot be named at all; yet this doesn’t stop us from making general assertions about the totality of real numbers—for example that every real number has a square, or that between any two distinct real numbers there is another that is distinct from either of them.

The quantifiers enable us to express general statements like these in the Predicate Calculus. Take any unary predicate $P(\_)$, simple or complex; then from it we can form two formulae $\forall x P(x)$ and $\exists x P(x)$. The former counts as true under an interpretion (D, I) so long as the extension of the predicate $P(\_)$ is the whole domain: so this formula asserts that *every* domain element has the property denoted by $P(\_)$. The latter is true under (D, I) so long as the extension of $P(\_)$ is non-empty: it asserts that *some* domain element has the property denoted by $P(\_)$. In these formulae the symbol x is a *variable*; it acts as a link between the quantifier and one or more argument places in the expression which follows. The usual convention is to write variables as lower-case letters from near the end of the alphabet, thereby distinguishing them from constants, which are written as lower-case letters from near the beginning of the alphabet.

Let us now see how we can use the Predicate Calculus to formalize inferences. Consider the following, rather curious inference:

Bob admires everyone who admires Carol
Carol admires herself
Therefore Bob admires himself

We shall use a binary predicate-letter A to represent 'admires', and constant terms b and c to stand for Bob and Carol respectively. To formalize the first premiss we note that what it says is that everyone has the following complex property: if they admire Carol, then Bob admires them. Hence we can formalize the inference as

$\forall x(A(x, c) \rightarrow A(b, x))$
$A(c, c)$
Therefore $A(b, b)$

Is this valid? We might argue the case as follows. The first premiss states that the property denoted by the complex predicate

$$A(\_, c) \rightarrow A(b, \_)$$

is possessed by every element in the domain. (Under our intended interpretation this is the property of either not admiring Carol or being admired by Bob.) In particular, it must, if the premiss is true, be possessed by the individuals denoted by c and b. This means that we can infer from the first premiss the two specific instances

$A(c, c) \rightarrow A(b, c)$
$A(b, c) \rightarrow A(b, b)$

The first of these, together with the second premiss ($A(c, c)$) implies $A(b, c)$ and this, together with the second formula above, implies $A(b, b)$, which is the desired conclusion.

The reasoning above was quite general: nothing in the way we argued depends for its soundness on the particular interpretation of the schematic letters we had in mind. The Predicate Calculus is adapted to precisely this kind of reasoning. A schematic inference is valid just so long as its conclusion comes out true in every interpretation which makes the premisses true. To demonstrate that an inference is valid we can either ask what an interpretation must be like if the premisses are to be true in it, and show that in any such interpretation the conclusion comes out true too (*model-theoretic* reasoning), or we can devise rules which operate on the premisses and conclusion of the inference itself, without reference to any interpretations (*proof-theoretic* reasoning). Model-theoretic reasoning has primacy over proof-theoretic in the sense that validity is ultimately defined in terms of interpretations, not in terms of rules of proof; but on the other hand model-theoretic reasoning tends to be rather intractable, since in general there are far too many different kinds of interpretations to consider. Hence we have good reason to be interested in proof-theoretic methods, just as with the Propositional Calculus, only more so, since whereas in the latter the number of relevantly different interpretations (i.e., truth-assignments) is always finite for any finite set of premisses, in the Predicate Calculus, there may be infinitely many distinct types of interpretation. In order to reason model-theoretically about them we cannot avoid devising rules of proof, which already takes us towards the development of a proof theory.

Proof theories for the Predicate Calculus can be obtained from Propositional Calculus proof theories by the addition of extra rules to cover the quantifiers, and the resulting systems can be shown to be both *sound* (that is, every inference which the system tells you is valid really is valid) and *complete* (that is, every inference which in fact is valid can be shown to be so using the system)[2].

3 First-order theories

The pure first-order Predicate Calculus described in the previous section has such a high degree of generality that it can be rather a cumbersome tool for practical inference. Many inferences which we make in everyday life and which are perfectly valid in an everyday sense become invalid if translated directly into the Predicate Calculus. An example is the simple inference

John is older than Mary
Therefore Mary is younger than John

This is certainly valid in the sense that the premiss can't be true without the conclusion being true as well. However, if we translate directly into Predicate Calculus notation we obtain the schematic inference

$P(a, b)$
Therefore $Q(b, a)$

which is invalid. The problem is that this inference is much more general that the original one. It allows informal interpretations such as

John admires Mary
Therefore Mary is fatter than John

which are plainly invalid. What has happened is that information has been lost in the transition from English to Predicate Calculus. The validity of our original inference depended on the *meanings* of the terms 'older' and 'younger' and not just on the logical form of the premiss and conclusion. In order to represent the inference successfully in the Predicate Calculus we have to represent the relevant features of the meanings of those terms in the form of extra premisses. In this case the relevant property is that if one person is older than another then the latter is younger than the former. By adding a premiss which captures this property we obtain the valid Predicate Calculus inference

$\forall x \forall y (P(x, y) \rightarrow Q(y, x))$
$P(a, b)$
Therefore $Q(b, a)$

[2]For the sake of completeness it must be mentioned here that the Predicate Calculus has been shown (by Church and Turing in 1936) to be *undecidable*: this means that there is no general algorithm for deciding whether or not an arbitrary inference is valid. Thus although we know that an inference is valid if and only if it can be proved to be valid using one of the standard sound and complete proof systems, there is no procedure by which we can be sure of telling whether it *can* be so proved. Admittedly, if it can be proved valid, then we can be sure of finding such a proof by means of a mechanical procedure (so we say that the Predicate Calculus is *semi-decidable*); the problem arises for inferences which are not valid—there is no procedure by which we can assure ourselves that there is no proof. What happens is this: there is a systematic procedure by which we can search for a proof, such that if a proof exists we can be sure of finding it. But if there is no such proof, then we may be condemned to searching for one forever!

Of course, if we are going to do a lot of reasoning involving 'older' and 'younger', then instead of adding the extra premisses to each inference it would be better to lay down appropriate formulae at the start as a fixed set of background assumptions which capture the logical properties of those terms. These background assumptions are called *axioms*, and they define a *first-order theory*. Using P and Q for 'older than' and 'younger than' respectively, as above, we might for example lay down the axioms:

(A1) $\forall x \neg P(x, x)$
(A2) $\forall x \forall y \forall z(P(x, y) \wedge P(y, z) \rightarrow P(x, z))$
(A3) $\forall x \forall y(Q(x, y) \leftrightarrow P(y, x))$

The first two axioms state that 'older than' is an irreflexive and transitive relation—in short, a strict partial order—and the third axiom states that 'younger than' is its converse. One can prove from these axioms that 'younger than' must be a strict partial order as well. In everyday terms, the axioms simply state that no-one is older than himself, that if one person is older than another who is older than a third then the first is older than the third, and that a person is younger than another just so long as the latter is older than the former. But this interpretation is only one of many possible interpretations under which our three axioms are satisfied: so that in general we may say that the axioms define not so much the first-order theory of 'older' and 'younger' as the first-order theory of an arbitrary strict partial order and its converse. It applies equally well to the relations $<$ and $>$ on the real numbers, or to the relations $\subset$ and $\supset$ on sets.

The axioms given above have two very desirable characteristics. First, they are *consistent*: this means that they do not contain an implicit (or explicit) contradiction, so that if we can infer a formula ϕ from them then we can't also infer $\neg\phi$. This is obviously desirable, since if we could infer both ϕ and $\neg\phi$ then there would be no interpretations which satisfy all the axioms, and so the axioms would be utterly useless. Second, the axioms are *complete*: suppose ϕ is any first-order formula containing the predicates P and Q such that ϕ is true of an arbitrary strict partial order when P is interpreted as the relation defining the order and Q is interpreted as its converse; then ϕ can be inferred from the axioms. In other words, what the completeness and consistency of axioms (A1)–(A3) amounts to is that they enable us to infer *all* and *only* those formulae in P and Q that are true of an arbitrary strict partial order when the predicates are interpreted in the way suggested.

If we want to apply the Predicate Calculus to some domain, what we have to do is to try to *axiomatize* the logical truths about that domain. Our goal is a consistent and complete axiomatization, though obviously if we can't achieve both consistency and completeness the former is more important than the latter. We can then reason about the domain by performing valid inferences in the first-order theory defined by our axiomatization.

An important first order theory is the Predicate Calculus with Identity. This is the theory of the *identity* relation, which holds between each object and itself but between no other pairs of objects. The usual symbol for this relation is the ordinary 'equals' sign, '=', used as an *infix*, i.e., it is placed *between* its arguments rather than before them (so we write '$a = b$' rather than '$=(a, b)$'). The axioms for equality are

$\forall x(x = x)$
$\forall x \forall y(\mathbf{\Phi}(x) \wedge x = y \rightarrow \mathbf{\Phi}(y))$

In the second axiom, the *substitution axiom*, the symbol $\mathbf{\Phi}$ can stand for any simple or complex predicate—so this is an *axiom schema*, standing in for an infinite set of substitution instances. For example, if we substitute '$\_ = x$' for $\mathbf{\Phi}(\_)$, we obtain the instance $\forall x \forall y(x = x \wedge x = y \rightarrow y = x)$, which, given the first axiom, reduces to $\forall x \forall y(x = y \rightarrow y = x)$ which states that identity is a symmetric relation. Again, if we put '$\_ = z$' for '$\Phi(\_)$' we obtain $\forall x \forall y \forall z(x = z \wedge x = y \rightarrow y = z)$, which asserts, in effect, that identity is a transitive relation[3].

The domain which above all others motivated the development of first-order logic was mathematics. Most branches of mathematics contain the arithmetic of the natural numbers as their core, so that was a prime target for axiomatization. The natural numbers can be generated from the base element 0 by repeated applications of the *successor* function s, which when applied to an arbitrary natural number will give us the next one in the series (thus the successor of 0 is 1, the successor of 1 is 2, and so on). The fundamental principle governing the natural numbers is *mathematical induction*, which in effect says that there aren't any natural numbers except those which you can reach by a finite number of applications of the successor function, beginning at 0. This principle is expressed in the axiom schema

$$(\Phi(0) \wedge \forall x(\Phi(x) \rightarrow \Phi(s(x)))) \rightarrow \forall x \Phi(x)$$

which says that if a predicate Φ applies to 0 and also applies to the successor of anything it applies to, then it applies to every natural number.

Addition and multiplication of natural numbers satisfy the following axioms

$$\forall x(x + 0 = x)$$
$$\forall x \forall y(x + s(y) = s(x + y))$$
$$\forall x(x \cdot 0 = 0)$$
$$\forall x \forall y(x \cdot s(y) = x \cdot y + x)$$

and everything that we know how to prove about addition and multiplication can be derived from these axioms together with the axioms for equality and the successor function. It is natural to suppose that all these axioms taken together should constitute a complete and consistent axiomatization of the arithmetic of the natural numbers, but it was shown by Kurt Gödel in 1931 that the natural numbers have such a rich structure that *no* such axiomatization is possible. The trouble is that arithmetic is rich enough for us to be able to use it to simulate any first-order axiom system by representing the language, rules of inference, and axioms of that theory as natural numbers, encoding the structure of each item by means of the factorization patterns of the numbers. In particular, given a consistent first-order axiomatization of arithmetic, then that system too could be simulated arithmetically, and what Gödel showed, by following through the details of such a simulation, was that one could construct a formula which expresses a true theorem of arithmetic but which could not be proved within the system, making the system incomplete. It follows that the axiom system we outlined above cannot be both complete and consistent: if it is consistent then it is not complete (that is, there are truths of arithmetic which it cannot be used to prove), while if it is complete then it is not consistent (so that we have no guarantee that anything we prove using the system is a truth of arithmetic).

Gödel's result shows us that formal logic has precisely definable limitations. This does not mean, however, that it is useless! Far from it: logical formalisms are of wide application in areas where it is possible to formulate sufficiently precisely what one wants to say. Such areas abound in the domain of computation and computer-related activities; an all-too-brief survey may be found in the sequel to this article.

[3] Note that we have 'smuggled in' an extra quantifier, to bind z. This reflects the fact that, for ease of understanding, we stated the substitution axiom in a slightly oversimplified form, omitting to mention that any extra variables in Φ need to be universally quantified in the axiom.

Acknowledgments

I should like to thank John Gooday, Brian Lings, and three anonymous reviewers for their helpful remarks on the first draft of this article.