CHAPTER 1

Introduction

Logic studies the relationship of implication between assumptions and conclusions. It tells us, for example, that the assumptions

> Bob likes logic. and
> Bob likes anyone who likes logic.

imply the conclusion

> Bob likes himself.

but not the conclusion

> Bob only likes people who like logic.

Logic is concerned not with the truth, falsity or acceptability of individual sentences, but with the relationships between them. If a conclusion is implied by true or otherwise acceptable assumptions, then logic leads us to accept the conclusion. But if an unacceptable or false conclusion is implied by given assumptions, then logic advises us to reject at least one of the assumptions. Thus, if I reject the conclusion that Bob likes himself then I am logically compelled to abandon either the assumption that Bob likes logic or the assumption that Bob likes anyone who likes logic.

To demonstrate that assumptions imply a conclusion, it is helpful to construct a proof consisting of inference steps. For the proof to be convincing, the individual inference steps need to be direct and obvious and should fit together correctly. For this purpose, it is necessary that the sentences be unambiguous and it is useful if the grammar of the sentences is as simple as possible. The requirement that the language of proofs be both unambiguous and grammatically simple motivates the use of a symbolic language rather than a natural language such as English.

The symbolic language of the clausal form of logic, used in the first nine chapters of this book, is exceedingly simple. The simplest sentences are <u>atomic</u> <u>sentences</u> which name relationships between individuals:

> Bob <u>likes</u> logic.
>
> John <u>likes</u> Mary.
>
> John <u>is</u> 2 <u>years older than</u> Mary.

(The underlined words are part of the names of relationships. Those not underlined are names of individuals.) More complex sentences express that atomic conditions imply atomic conclusions:

Mary likes John if John likes Mary.

Bob likes x if x likes logic.

Here x is a variable which names any individual. Sentences can have several joint conditions or several alternative conclusions:

Mary likes John or Mary likes Bob if Mary likes x.
(Mary likes John or Bob if she likes anything at all).

x likes Bob if x is a student of Bob and x likes logic.

Sentences are also called clauses. In general, every clause expresses that a number (possibly zero) of joint conditions imply a number (possibly zero) of alternative conclusions. Conditions and conclusions express relationships among individuals. The individuals may be fixed and named by words such as

Bob, John, logic or 2

called (somewhat confusingly, perhaps) constant symbols, or they may be arbitrary and named by variables such as

u, v, w, x, y, z.

The use of function symbols to construct more complex names such as

dad(John) (i.e. John's dad)

fraction(3,4) (i.e. the fraction 3/4)

will be considered later.

This informal outline of the clausal form of logic will be elaborated and slightly modified in the next section of this chapter. But the great simplicity of clausal form compared with natural languages should already be apparent. It is surprising therefore that clausal form has much of the expressive power of natural language. In the last four chapters of the book we shall investigate some of the shortcomings of clausal form and propose ways of overcoming them.

The family relationships example and clausal form

It is convenient to express the atomic formulae which serve as the conditions and conclusions of clauses in a simplified, if somewhat less natural, form. The name of the relation is written in front of the atomic formula, followed by the sequence of names of individuals to which the relation applies. Thus we write Father(Zeus,Ares) instead of Zeus is father of Ares and Fairy-Princess(Harmonia) instead of Harmonia is a fairy princess. Here, strictly speaking, "Fairy-princess" names a property of individuals rather than a relation among individuals. However, in order to simplify the terminology, we shall include properties (also called predicates) when we speak of relations.

Moreover, to mix terminology thoroughly we shall refer to names of relations as predicate symbols.

We use the arrow <-, read "if", to indicate implication, writing, for example,

Female(x) <- Mother(x,y)

to express that

x is female if x is mother of y.

To simplify notation and the inference rules later on, it is convenient to regard all clauses as implications, even if they have no conditions or conclusions. Thus we write

Father(Zeus,Ares) <-

instead of

Father(Zeus,Ares).

Implications without conclusions are denials. The clause

<- Female(Zeus)

expresses that Zeus is not female.

The following clauses describe some of the properties and family relationships of the Greek gods.

F1 Father(Zeus,Ares) <-

F2 Mother(Hera,Ares) <-

F3 Father(Ares,Harmonia) <-

F4 Mother(Aphrodite,Harmonia) <-

F5 Father(Cadmus,Semele) <-

F6 Mother(Harmonia,Semele) <-

F7 Father(Zeus,Dionysus) <-

F8 Mother(Semele,Dionysus) <-

F9 God(Zeus) <-

F10 God(Hera) <-

F11 God(Ares) <-

F12 God(Aphrodite) <-

F13 Fairy-Princess(Harmonia) <-

The intended meaning of the clauses should be obvious. The following clauses constrain, and therefore help to clarify, their meaning.

F14 Female(x) <- Mother(x,y)

F15 Male(x) <- Father(x,y)

F16 Parent(x,y) <- Mother(x,y)

F17 Parent(x,y) <- Father(x,y)

These clauses state that, for all x and y,

x is female if x is mother of y,

x is male if x is father of y,

x is parent of y if x is mother of y, and

x is parent of y if x is father of y.

Variables in different clauses are distinct even if they have the same name. Thus the variable x in clause F14 has no connection with the variable x in F15. The name of a variable has significance only within the context of the clause in which it occurs. Two clauses which differ only in the names of the variables they contain are equivalent and are said to be *variants* of one another.

In the clausal form, all the conditions of a clause are conjoined together (i.e. connected by "and"), whereas all the conclusions are disjoined (i.e. connected by "or"). Hence the connectives "and" and "or" can safely be replaced by commas. Commas between conditions, therefore, are read as "and" and between conclusions are read as "or". Thus

F18 Grandparent(x,y) <- Parent(x,z), Parent(z,y)

F19 Male(x), Female(x) <- Human(x)

where x, y and z are variables, state that for all x, y and z

x is grandparent of y if x is parent of z *and*
z is parent of y,

x is male *or* x is female if x is human.

If several conclusions are implied by the same conditions then separate clauses are needed for each conclusion. Similarly if the same conclusion is implied by alternative conditions then separate clauses are needed for each condition. For example, the sentence

Female(x) *and* Parent(x,y) <- Mother(x,y)

which can be expressed directly in the standard form of logic (defined in Chapter 10) can be expressed equivalently by the clauses

Female(x) <- Mother(x,y)

Parent(x,y) <- Mother(x,y).

The two clauses are implicitly connected by "and": i.e. x is female if x is the mother of y and x is the parent of y if x is the mother of y. Similarly, the sentence

Parent(x,y) <- Mother(x,y) or Father(x,y)

can be expressed by the clauses

Parent(x,y) <- Mother(x,y)

Parent(x,y) <- Father(x,y)

x is parent of y if x is mother of y and
x is parent of y if x is father y.

Predicate symbols can name relationships among more than two individuals. For example, the atomic formula

Parents(x,y,z)

could be used to express that

x is the father of z and y is the mother of z

i.e. Parents(x,y,z) <- Father(x,z), Mother(y,z).

A more precise definition of clausal form

We shall define the syntax (grammar) of clausal form more precisely and at the same time indicate its correspondence with English.

> A clause is an expression of the form
>
> $B_1,\ldots,B_m \leftarrow A_1,\ldots,A_n$
>
> where $B_1,\ldots,B_m,A_1,\ldots,A_n$ are atomic formulae, $n \geq 0$ and $m \geq 0$. The atomic formulae $A_1,\ldots,A_n$ are the joint conditions of the clause and $B_1,\ldots,B_m$ are the alternative conclusions. If the clause contains the variables $x_1,\ldots,x_k$ then interpret it as stating that
>
> for all $x_1,\ldots,x_k$
> B_1 or ... or B_m if A_1 and ... and A_n.
>
> If $n = 0$ then interpret it as stating unconditionally that
>
> for all $x_1,\ldots,x_k$
> B_1 or ... or B_m.

If $m = 0$ then interpret it as stating that

> for all $x_1,\ldots,x_k$
> it is not the case that
> A_1 and ... and A_n.

If $l = n = 0$ then write it as □ and interpret it as a sentence which is always false.

An *atom* (or *atomic formula*) is an expression of the form

$$P(t_1,\ldots,t_m)$$

where P is an m-place predicate symbol, $t_1,\ldots,t_m$ are terms and $m \geq 1$. Interpret the atom as asserting that the relation called P holds among the individuals called $t_1,\ldots,t_m$.

A *term* is a variable, a constant symbol or an expression of the form

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

where f is an m-place function symbol, $t_1,\ldots,t_m$ are terms and $m \geq 1$.

The sets of *predicate symbols*, *function symbols*, *constant symbols* and *variables* are any mutually disjoint sets. By convention, we reserve the lower case letters

$$u, v, w, x, y, z,$$

with or without adornments, for variables. The types of other kinds of symbols can be identified by the positions they occupy in clauses.

The arrow of clausal form <- is written in the opposite direction to that normally used in the standard form of logic. Where we write

B <- A (B if A)

it is more usual to write

A -> B (if A then B).

The difference, however, is only superficial. We use the notation B <- A in order to draw attention to the conclusion of the clause.

The various places of a predicate symbol or function symbol are also called its *arguments*. In the atom $P(t_1,\ldots,t_m)$, the first argument is t_1 and the last argument is t_m.

Composite terms are needed in order to refer to infinitely many individuals using only finitely many clauses. For example, the non-negative integers can be represented by the terms

$$0,\ s(0),\ s(s(0)),\ \ldots,\ \underbrace{s(s(\ldots s(0)\ldots))}_{n \text{ times}},\ \ldots$$

where Ø is a constant symbol and s is a 1-place function symbol (s stands for "successor"). The term s(t) names the number which is one larger then the number named by the term t. It is the successor of t in the succession of integers. The clauses

Num1 Numb(Ø) <-

Num2 Numb(s(x)) <- Numb(x)

state that

Ø is a number and

s(x) is a number if x is.

Top-down and bottom-up presentation of definitions

The definition of clausal form has been presented in a top-down manner. The first definition explains the goal concept of clause in terms of the concept of atomic formula, (which has not yet been defined). It becomes the new goal concept, which in the next definition is reduced to the two subgoal concepts of predicate symbol and term. The concept of term is defined recursively and reduces eventually to the concepts of constant symbol, variable and function symbol. Thus the original concept finally reduces to the four concepts of predicate symbol, constant symbol, variable and function symbol. It does not matter what objects these symbols are, provided they can be distinguished from one another and do not get confused with the "reserved" symbols:

<- , (and)

We assume therefore that the reserved symbols are not contained within the other symbols.

The top-down presentation of definitions has the advantage of always being well-motivated. Its disadvantage is that, since goal concepts are defined in terms of subgoal concepts which are not yet defined, definitions cannot be completely understood as they are presented.

The bottom-up presentation of definitions is the opposite. It begins with concepts which are undefined, either because they are "primitive" and undefinable or else because they are already well understood. Then it defines new concepts in terms of ones already given. The definitions terminate when the goal concept has been defined. Definitions can be understood as soon as they are given, but the motivation cannot be appreciated until all the definitions have been completed.

The distinction between top-down and bottom-up applies not only to the presentation of definitions, but also to the presentation and discovery of proofs and to the writing of computer programs. Proofs can be presented in the traditional, bottom-up, mathematical manner; reasoning forward from what is given, deriving new conclusions from previous ones and terminating when the goal has been derived. Alternatively, proofs can be presented in a top-down manner which reflects the process of their discovery; reasoning backward from the goal, by reducing goals to subgoals and terminating when all the subgoals are recognised as solvable.

Computer programs also can be written bottom-up, starting with primitive programs already understood by the computer and writing new programs in terms of old ones. At each stage the programs can be executed by the computer and can be tested. If the low-level programs already written cannot be put together into suitable higher-level programs, then they have to be rewritten. Experience teaches that it is better to write programs top-down, writing the highest-level programs first in terms of unwritten lower-level ones. The lower-level programs are written later and are guaranteed to fit together properly. Moreover, the lower-level programs later can be changed and improved without affecting the rest of the program.

Together with the utility of using symbolic logic to represent information, the distinction between top-down and bottom-up reasoning is one of the major themes of this book. It is the distinction between analysis (top-down) and synthesis (bottom-up), between teleology (top-down) and determinism (bottom-up). Moreover, the use of top-down inference in preference to bottom-up inference reconciles the classical, logical view of reasoning as it <u>ought to be performed</u> with the psychological view of reasoning as it <u>is performed</u> by human beings in practice.

Top-down reasoning relates the human problem-solving strategy of reducing goals to subgoals to the method of executing computer programs by replacing procedure calls with procedure bodies. It unifies the study of logic with both the study of human problem-solving and the study of computer programming.

<u>Semantics of clausal form</u>

<u>Syntax</u> deals with the grammar of sentences. Historically, it also deals with inference rules and proofs. <u>Semantics</u>, on the other hand, deals with meaning. The translation of clauses into English gives only an informal guide to their semantics.

In natural languages we speak casually of words and sentences as having meanings. In symbolic logic we are more careful. Any meaning that might be associated with a predicate symbol, constant symbol, function symbol or sentence is relative to the collection of sentences which express all the relevant assumptions. In the family relationships example, for instance, if F1-19 express all the assumptions, then there is nothing to rule out an interpretation in which the assertion

F Mother(Zeus,Ares) <-

holds. Such a possibility is consistent with the stated assumptions F1-19, which alone determine any meaning that might be associated with the symbols

"Mother", "Father", "Zeus", etc.

To rule out the possibility F we need some additional assumption such as

F20 <- Male(x), Female(x).

F is consistent with F1-19 but inconsistent with F1-20.

Given a set of clauses which express all the assumptions concerning a problem-domain, to understand any individual symbol or clause it is necessary to determine what is logically implied by the assumptions. The meaning of a predicate symbol, such as "Mother", might be identified with the collection of all sentences which contain the predicate symbol and are logically implied by the assumptions. Thus the meaning of "Mother" in F1-20 includes the denial

F* <- Mother(Zeus,Ares)

but the meaning of "Mother" in F1-19 does not.

It follows that it is unnecessary to talk about meaning at all. All talk about meaning can be reexpressed in terms of logical implication. To define the semantics of the clausal form of logic, therefore, it suffices to define the notion of logical implication.

In the clausal form of logic, to determine that a set of assumptions imply a conclusion we deny that the conclusion holds and show that the denial of the conclusion is inconsistent with the assumptions. The semantics of clausal form, therefore, reduces to the notion of inconsistency. To determine, for example, that the consequence F* is part of the meaning of motherhood as determined by the clauses F1-20, we show that the denial of F*, namely the assertion F, is inconsistent with F1-20. The reduction of semantics to the notion of inconsistency may seem unnatural, but it has significant computational advantages.

The inconsistency of a set of clauses can be demonstrated "semantically" by showing that no interpretation of the set of clauses makes them all true, or it can be demonstrated "syntactically" by constructing a proof consisting of inference steps. This book is about the syntactic, proof-theoretic method of demonstrating inconsistency. But, because clauses can be understood informally by translating them into English or more formally by considering the interpretations in which they are true, we shall delay the investigation of inference rules and proofs until Chapter 3.

The semantics of symbolic logic, based upon the notion of interpretation, is independent of the inference rules used to manipulate expressions in the language. This distinguishes logic from the vast majority of formalisms employed in computing and artificial intelligence. Programs expressed in normal programming languages need to be understood in terms of the behaviour they evoke inside a computer. The burden of communication falls upon the programmer, who needs to express information in machine-oriented terms. However, when programs are expressed in symbolic logic, they can be understood in terms of their human-oriented, natural language equivalents. The burden of communication then falls upon the machine, which needs to perform mechanical operations (equivalent to inference steps) to determine whether the information expressed in a program logically implies the existence of a solution to a given problem. The machine needs to be a problem-solver. The tasks of constructing proofs, executing programs and solving problems become identical. Moreover, similar problem-solving strategies apply, whether they are applied by human-beings to problems posed in natural language or by machines to problems posed in symbolic logic.

Before presenting the precise, semantic definitions of inconsistency and interpretation, we shall illustrate by examples some of the expressive capabilities of clausal form and some of the characteristics of its semantics.

The fallible Greek example

To show that the assumptions

G1 Human(Turing) <-

G2 Human(Socrates) <-

G3 Greek(Socrates) <-

G4 Fallible(x) <- Human(x)

imply the conclusion that there is a fallible Greek, we deny the conclusion

G5 <- Fallible(u), Greek(u)

and show that the resulting set of clauses is inconsistent. Moreover, the demonstration of inconsistency can be analysed to determine the reason for the inconsistency of G5 with G1-4, namely the substitution

u = Socrates

which identifies an individual that is both fallible and Greek. In this way the clause G5 can be regarded as expressing the problem of finding an individual u which is a fallible Greek. The substitution, u = Socrates, which can be extracted from the proof, can be regarded as a solution to the problem.

The example of the fallible Greek was first introduced to explain the behaviour of programs written in the programming language PLANNER [Hewitt 1969]. Our intention here is just the opposite: to show that information expressed in logic can be understood without understanding the behaviour it evokes inside a machine.

The factorial example

The fallible Greek example is not typical of programs written in conventional programming languages. However, the factorial example is.

The factorial of 0 is 1.
The factorial of x+1 is x+1 times the factorial of x.

The simplest formulation of the definition uses function symbols:

```
fact(x)     names   the factorial of x,
times(x,y)          the product of x and y,
s(x)                x+1.
```

A 2-place predicate symbol expresses equality. Equal(x,y) holds when x "is" y.

```
Equal(fact(Ø), 1) <-

Equal(fact(s(x)), times(s(x), fact(x))) <-
```

To complete the definition, additional definitions are needed to characterise "times" and "Equal". The following clauses are typical of the ones which are necessary for equality.

```
(1)           Equal(x,x) <-

(2)           Equal(x,y) <- Equal(x,z), Equal(z,y)

(3)           Equal(fact(x), fact(y)) <- Equal(x,y)
```

To find the factorial of 2, for example, we deny that it exists:

```
(4)           <- Equal(fact(s(s(Ø))), w)
```

But (1) and (4) alone are inconsistent and the substitution

```
w = fact(s(s(Ø)))
```

can be identified as the reason for inconsistency. Unfortunately, the substitution is not very informative.

The problem is that the function symbols "fact", "times" and "s" allow numbers to be referred to by many different names. The variable-free terms

```
s(s(Ø)),  s(1),  s(fact(Ø)),  s(fact(times(Ø, s(Ø))))
```

all name the same number 2 and are equal to one another. The problem can be solved if individuals are given unique names. In this example it suffices to employ only the constant symbol Ø and the function symbol s. The factorial and multiplication functions can be treated as relations.

```
Fact(x,y)     holds when the factorial of x is y.
Times(x,y,z)  holds when x times y is z.
```

Then the clauses

```
Fact1         Fact(Ø, s(Ø)) <-

Fact2         Fact(s(x), u) <- Fact(x,v), Times(s(x), v, u)
```

completely define the factorial relationship relative to an appropriate definition of multiplication. The equality relation does not appear and its definition is unnecessary. Assume that a definition of multiplication, including such clauses as

```
Times(Ø,x,Ø) <-

Times(s(Ø), y, y) <-

       etc.
```

is provided. To solve the problem of finding the factorial of 2, we deny that it exists.

```
Fact3          <- Fact(s(s(Ø)), w)
```

The resulting set of clauses Fact1-3 is inconsistent with any definition of Times which implies the assertions

```
Times(s(s(Ø)), s(Ø), s(s(Ø))) <-

Times(s(Ø), s(Ø), s(Ø)) <- .
```

Given a demonstration of inconsistency it is possible to extract the only substitution

```
w = s(s(Ø))
```

which solves the problem. In this way the definition of Fact supplemented by a definition of Times serves as a program which can be used by a computer to calculate factorials. The program can be understood without understanding how the computer works.

The universe of discourse and interpretations

In this section and the next we define the semantics of clausal form. These sections are more rigorous than the rest of the chapter and may be safely skimmed through on a first reading.

The two formulations of the factorial definition illustrate a general principle of clausal writing style. To avoid problems associated with individuals having more than one name, constant symbols and function symbols should be used sparingly. If individuals are named by unique variable-free terms, then the <u>universe of discourse</u> of a set of clauses, which intuitively represents the collection of all individuals described by the clauses, can be identified with the collection of all variable-free terms which can be constructed from the constant symbols and function symbols occurring in the set of clauses. A candidate <u>interpretation</u> for a set of clauses can then be regarded as any assignment to each n-place predicate symbol occurring in the set of clauses of an n-place relation over the universe of discourse.

The assumptions G1-4 of the fallible Greek problem are a simple example. They have a small, finite universe of discourse, consisting of the two constant symbols

"Turing" and "Socrates".

To specify a candidate interpretation is to specify a relation over the universe of discourse for each of the three predicate symbols in the set of clauses. Each predicate symbol can be assigned four different interpretations and therefore the set of clauses as a whole has a total of

4*4*4 = 64

different candidate interpretations.* But only two of them make all of the clauses G1-4 true. One of them makes all of the variable-free atoms

Human(Socrates), Human(Turing),
Fallible(Socrates), Fallible(Turing),
Greek(Socrates), Greek(Turing)

true. The other makes the atoms

Human(Socrates), Human(Turing),
Fallible(Socrates), Fallible(Turing),
Greek(Socrates)

true but Greek(Turing)

false.

The larger set of clauses G1-5 has the same universe of discourse and the same collection of 64 candidate interpretations. However, none of the 64 interpretations make all five clauses G1-5 simultaneously true. The two interpretations which make G1-4 all true make G5 false. In particular the instance

G'5 <— Fallible(Socrates), Greek(Socrates)

of G5, in which u = Socrates, is false in both interpretations, because the two conditions

Fallible(Socrates) and Greek(Socrates)

denied by G'5 are true in both interpretations. Since G'5 is false in both interpretations, G5 is false also (because a clause containing variables is true in an interpretation if and only if all its instances are true and is false if one of its instances is false). Therefore G1-5 is inconsistent because there is no interpretation which makes all of its clauses true. By analysing the proof of inconsistency it is possible to identify the individual

u = Socrates

whose existence is inconsistently denied by the clause G5.

The semantic method of showing the inconsistency of a set of clauses, by demonstrating that no interpretation makes all of its clauses true, is a general method which can be used for any set of clauses. Moreover,

---

* The symbol "*" is used throughout this book for multiplication.

the interpretations which need to be considered can always be restricted to those whose domain of individuals consists of the universe of discourse. If the set of clauses contains no constant symbols, then it is necessary to include in the universe of discourse a single, arbitrary constant symbol. In this case *the universe of discourse* consists of all variable-free terms which can be constructed from the given constant symbol symbol and any function symbols which might occur in the set of clauses.

The inclusion of an arbitrary constant symbol in the universe of discourse, if there is none in the set of clauses, formalises the assumption that at least one individual exists. Because of this assumption, the clause

(1) Good(x) <-

which expresses that everything is good, implies that at least one thing is good. It is inconsistent with the assumption that nothing is good

(2) <- Good(y) .

The universe of discourse consists of some single, arbitrary constant symbol, say ◇. There are only two candidate interpretations - one in which

Good(◇) is true

the other in which

Good(◇) is false.

The first interpretation falsifies (2). The second interpretation falsifies (1). So (1) and (2) are, therefore, simultaneously true in no interpretation and are inconsistent. Notice that the demonstration of inconsistency does not depend on the name of the arbitrary member of the universe of discourse. The argument is the same no matter what constant symbol is used.

The notion of interpretation itself can be simplified. To specify an interpretation it suffices to specify its effect on the truth or falsity of variable-free atomic formulae. An *interpretation* of a set of clauses, therefore, can be regarded as any assignment of either one of the two *truth values*

true or *false*

to every every variable-free atom which can be constructed from the universe of discourse and the predicate symbols occurring in the set of clauses.

A more precise definition of inconsistency

We are now in a position to present a more precise definition of inconsistency.

> A set of clauses S is <u>inconsistent</u> if and only if it is not consistent. It is consistent if and only if all its clauses are true in some interpretation of S.
>
> A <u>clause is true</u> in an interpretation of a set of clauses S if and only if every variable-free instance of the clause, obtained by replacing variables by terms from the universe of discourse of S, is true in the interpretation. Otherwise the <u>clause is false</u> in the interpretation.
>
> A <u>variable-free clause is true</u> in an interpretation I if and only if whenever all of its conditions are true in I, at least one of its conclusions is true in I. Equivalently, the clause is true in I if and only if at least one of its conditions is false in I or at least one of its conclusions is true in I. Otherwise, the <u>clause is false</u> in I.

The precise definition of inconsistency clarifies the semantics of the empty clause, □. Since the empty clause has neither conditions nor conclusions it cannot possibly be true in any interpretation. It is the only clause which is self-inconsistent. To demonstrate the inconsistency of a set of clauses it suffices to demonstrate that it logically implies the obviously inconsistent empty clause. The empty set of clauses, however, is consistent. All clauses which belong to it are true in all interpretations, since it contains no clauses which can be false.

The notions of instantiation and substitution are important not only for defining the semantics of clausal form but also for defining the inference rules later on. An <u>instance</u> of a clause is obtained by applying a substitution to the clause. A <u>substitution</u> is an assignment of terms to variables. Only one term is assigned to any given variable. It is convenient to represent a substitution as a collection of independent substitution components:

$$\{x_1 = t_1, \quad x_2 = t_2, \quad \ldots., \quad x_m = t_m\}$$

Each component $x_i = t_i$ of the substitution assigns a term t_i to a variable x_i. The <u>result of applying a substitution</u> σ to an expression E is a new expression $E\sigma$ which is just like E except that, wherever σ contains a substitution component $x_i = t_i$ and E contains an occurrence of the variable x_i, the new expression contains an occurrence of t_i. The application of σ to E replaces all occurrences of the same variable by the same term. The <u>expression</u> E can be any term, atom, clause or set of clauses. Different variables may be replaced by the same term.

It follows that distinct variables do not necessarily refer to distinct individuals. The assumptions

L1 Likes(Bob,logic) <-

L2 Likes(Bob,x) <- Likes(x,logic)

L3 <- Likes(x,y), Likes(y,y)
No one likes anyone who likes himself.

for example, are inconsistent because L1 and L2 are inconsistent with the instance

<- Likes(Bob,Bob), Likes(Bob,Bob)

of L3 in which both x = Bob and y = Bob.

The semantics of alternative conclusions

The precise definition of inconsistency clarifies the semantics of alternative conclusions. If a clause has several conclusions, then it should be interpreted as stating that if all its conditions hold then at least one (but possibly more) of its conclusions hold. This inclusive interpretation of "or" contrasts with the exclusive interpretation in which "A or B" is interpreted as expressing that either one or other of A and B holds, but not both.

The inclusive interpretation of "or" implies, for example, that the set of assumptions

B1 Animal(x), Mineral(x), Vegetable(x) <-

B2 Animal(x) <- Oyster(x)

B3 Mineral(x) <- Brick(x)

B4 Vegetable(x) <- Cabbage(x)

is consistent with the possibility that something is both an animal and a vegetable:

B5 Animal(x) <- Bacterium(x)

B6 Vegetable(x) <- Bacterium(x)

B7 Bacterium(⊙✓) <-

The exclusive sense of "or" can be captured by means of inclusive "or" and denial. To express, for example, that every human is either male or female but not both, requires two clauses:

Female(x), Male(x) <- Human(x)

<- Female(x), Male(x), Human(x)

Horn clauses

For many applications of logic, it is sufficient to restrict the form of clauses to those containing at most one conclusion. Clauses containing at most one conclusion are called Horn clauses, because they were first investigated by the logician Alfred Horn [1951]. It can be shown, in fact, (exercise 5 in Chapter 12) that any problem which can be expressed in logic can be reexpressed by means of Horn clauses.

The majority of formalisms for computer programming bear greater resemblence to Horn clauses than they do to "non-Horn" clauses. In addition, most of the models of problem-solving which have been developed in artificial intelligence can be regarded as models for problems expressed by means of Horn clauses.

Because Horn clauses are such an important subset of clausal form, and because inference methods for Horn clauses have a simple problem-solving and computer programming interpretation, we shall investigate them in detail (in Chapters 3-6) before investigating the full clausal form in general (in Chapters 7-8). It is important to appreciate, however, that although non-Horn clauses might be dispensible in theory they are indispensible in practice. Moreover, the extension of Horn clause problem-solving methods to clausal form in general is a significant extension of the simpler models of problem-solving which are more popular today.

Mushrooms and toadstools

A simple example which can be expressed naturally only by means of non-Horn clauses is one which expresses some typical beliefs concerning mushrooms and toadstools. Suppose I believe

(1) Every fungus is a mushroom or a toadstool.

(2) Every boletus is a fungus.

(3) All toadstools are poisonous. and

(4) No boletus is a mushroom.

Symbolically,

Fung1 Mushroom(x), Toadstool(x) <- Fungus(x)

Fung2 Fungus(x) <- Boletus(x)

Fung3 Poisonous(x) <- Toadstool(x)

Fung4 <- Boletus(x), Mushroom(x)

then I should also believe at least the more obvious of the logical consequences of my beliefs. In particular I should believe that

All boleti are poisonous.

Fung5 Poisonous(x) <- Boletus(x)

But every collector of edible fungi knows that few boleti are poisonous and most are quite tasty. If I reject the conclusion Fung5 and maintain my belief in logic then I must reject at least one of my initial assumptions Fung1-4. It is surprising how many people abandon logic instead.

Exercises

1) Using the same vocabulary (i.e. predicate symbols, constants and function symbols) as in F1-19, express the following sentences in clausal form:

a) x is a mother of y if
x is a female and x is a parent of y.

b) x is a father of y if
x is a male parent of y.

c) x is human if
y is a parent of x and y is human.

d) An individual is human if
his (or her) mother is human and
his (or her) father is human.

e) If a person is human
then his (or her) mother is human or
his (or her) father is human.

f) No one is his (or her) own parent.

2) Given clauses which define the relationships

Father(x,y)	(x is father of y)
Mother(x,y)	(x is mother of y)
Male(x)	(x is male)
Female(x)	(x is female)
Parent(x,y)	(x is parent of y)
Diff(x,y)	(x is different from y)

define the following additional relationships:

M(x)	(x is a mother)
F(x)	(x is a father)
S(x,y)	(x is a son of y)
D(x,y)	(x is a daughter of y)
Gf(x,y)	(x is a grandfather of y)
Sib(x,y)	(x is a sibling of y)

For example the clause

Aunt(x,y) <- Female(x), Sib(x,z), Parent(z,y)

defines the relationship Aunt(x,y) (x is an aunt of y) in terms of the Female, Sib and Parent relations.

3) Let the intended interpretation of

Hc(x)	be	x is a heavenly creature
Wd(x)		x is worth discussing
Star(x)		x is a star
Comet(x)		x is a comet
Planet(x)		x is a planet
Near(x,y)		x is near y
Ht(x)		x has a tail.

a) Express in clausal form the assumptions:

Every heavenly creature worth discussing is a star, planet or comet.
Venus is a heavenly creature, which is not a star.
Comets near the sun have tails.
Venus is near the sun but does not have a tail.

b) What "obvious" missing assumption needs to be added to the clauses above for them to imply the conclusion

Venus is a planet ?

4) Using only the predicate symbols, Numb, Odd and Even, the function symbol s, and the constant Ø, express in clausal form

a) the conditions under which a number is even,

b) the conditions under which a number is odd,

c) that no number is both odd and even,

d) that a number is odd if its successor is even,

e) that a number is even if its successor is odd,

f) that the successor of a number is odd if the number is even and that the successor is even if the number is odd.

5) Let the intended interpretation of

Parity(x,odd)	be x is odd
Parity(x,even)	be x is even.

Let the notion of opposite parities be expressed by the two clauses

```
Opp(odd,even) <-
Opp(even,odd) <-
```

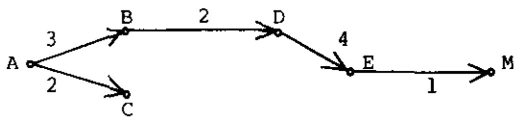
Define the notion of a number being odd or even using only three additional clauses, two of them variable-free assertions.

6) Inventing your own predicate symbols, express the following assumptions in clausal form. Use only two constants, one to name my cat, the other to name me.

Birds like worms.
Cats like fish.
Friends like each other.
My cat is my friend.
My cat eats everything it likes.

What do these assumptions imply that my cat eats?

7) Assume that arcs in a directed graph, e.g.



are described by assertions of the form

Distance(r,s,t) <-
(the length of the arch from r to s is t).

Thus the assertion

Distance(A,B,3) <-

describes the arc from A to B. Assume also that the relationship

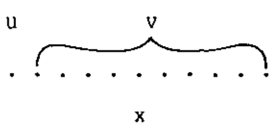
Plus(x,y,z),

which holds when x+y = z, is already given. Using only one clause, extend the definition of the relationship Dist(x,y,z) so that it expresses that there is a path of length z from x to y.

8) Assume that the relationships

Empty (x)	(the list x is empty)
First(x,u)	(the first element of list x is u)
Rest(x,v)	(the rest of the list x following the first element, is the list v)

are already given. Pictorially, the relationship



holds when both of the conditions First(x,u) and Rest(x,v) hold.

a) Define the new relationship

Memb(z,x) (element z is a member of list x)

in terms of the First and Rest relations. Two clauses are necessary.

b) Define the relationship

Sub(x,y) (all elements of list x are elements of list y)

in terms of the Empty, First, Rest and Memb relations.

c) Assume

Plus(x,y,z) (x + y = z)

is given. Define the relationship

Sum(x,w) (the sum of all elements in the list of numbers x is w)

in terms of the Empty, First, Rest and Plus relations.

9) Using predicate symbols of your own invention, but no function symbols or constants, express the following sentences in clausal form:

No dragon who lives in a zoo is happy.
Any animal who meets kind people is happy.
People who visit zoos are kind.
Animals who live in zoos meet the people who visit zoos.

What two missing additional assumptions are needed to justify the conclusion

No dragon lives in a zoo. ?

10) There are four different variable-free atoms which can be constructed from the vocabulary of clauses L1-3. Consequently there are 16 different interpretations of L1-3. How many of these interpretations make both L1 and L2 true? How many make L3 true? How many make all of L1-3 true?