

CHAPTER 2

Representation in Clausal Form

In order to construct a mechanical problem-solving system, it is necessary to express information in an unambiguous language. Moreover, for the system also to serve as a model of human problem-solving, the language needs to resemble the natural languages used by human beings. The language of symbolic logic is both precise enough to be understood and manipulated by computers and natural enough to be regarded as a simplified form of natural language.

In this chapter, we shall compare the clausal form of logic with some of the features of natural language. We shall also compare it with semantic networks for representing natural language meanings and with relational databases for representing information in computers. In order to make the relationship between logic and natural language more apparent, we introduce the infix notation for predicate symbols.

Infix notation

The informal notation used to introduce clausal form at the beginning of the first chapter can be given formal status.

Binary (two-place) predicate symbols can be written between their arguments. Instead of writing atoms in prefix form

$$= (x,y), \leq (x,y), \text{Father}(x,y)$$

we can write them in infix form

$$x = y, \quad x \leq y, \quad x \text{ is the father of } y$$

respectively. The expression "is the father of" is regarded as a single predicate symbol.

Unary (one-place) predicate symbols can be written after their arguments, without the attendant parentheses. Thus we can write

$$x \text{ is good} \leftarrow x \text{ accomplishes } y, y \text{ is good}$$

instead of

$$\text{Good}(x) \leftarrow \text{Accomplishes}(x,y), \text{Good}(y).$$

Unary predicate symbols written after their arguments are also regarded

as infix notation.

For predicate symbols having more than two arguments, infix notation distributes parts of a predicate symbol between its arguments. Thus we can write

John gave book to Mary <-

instead of

Gave(John, book, Mary) <-

where "gave" and "to" are regarded as the first and second parts of the single predicate symbol "Gave".

Infix notation, though easier to read, increases the possibility of ambiguity. The expression

John is a student <-

in infix notation can be interpreted as either one of the two clauses

Student(John) <-

Isa(John, student) <-

in prefix notation. To eliminate ambiguity, we underline infix predicate symbols and their parts. Thus the atom in the clause

John is a student <-

has one argument, whereas the atom in

John is a student <-

has two arguments. Underlining may be omitted, as in the case of the two binary predicate symbols "=" and "≤", when there is no ambiguity.

Infix notation can also be employed for function symbols. We can write

$x + y$, $x * y$, $x!$, $x + 1$, x 's dad

for example, instead of

$+(x,y)$, $\text{times}(x,y)$, $\text{fact}(x)$, $s(x)$, $\text{dad}(x)$.

Infix notation for function symbols and associated conventions for reducing parentheses will be discussed again in Chapter 5.

Variables and types of individuals

The analogue of variables in logic are such words in English as

"something", "anything", "everything",
"nothing", "a thing", "things".

For example,

```
<- x is good, x is bad
Nothing is both good and bad.

x is bad <- x accomplishes y, y is bad
Anything which accomplishes something bad is bad itself.
```

There are many occasions, however, in which logic uses a variable, but English uses a word which refers to a specific type (or classification) of individual. It is usual in logic to name types by means of one-argument predicate symbols. Thus, the English sentence

All men are animals.

would be expressed by the clause

```
x is an animal <- x is a man .
```

The variable x in the clause is avoided in the English by referring to the type "men". This is even more obvious if the English sentence is paraphrased

Men are animals.

The English words "anyone", "everyone", "anywhere", "somewhere", "anytime", "sometime" refer to individuals of type "human", "place", and "time".

Relative pronouns in English, such as "who", "which" and "where" refer to individuals already mentioned in the same sentence. For example

```
Anyone who eats animals is a carnivore.
x is a carnivore <- x is human,
                     x eats y,
                     y is an animal
```

The restrictive relative clause

who eats animals

adds two extra conditions concerning the individual x mentioned in the main sentence

```
Anyone is a carnivore.
x is a carnivore <- x is human
```

The non-restrictive relative clause, however, in the sentence

```
John, who eats animals, is a carnivore.
John is a carnivore <-
John eats y <- y is an animal
```

adds an extra sentence to the main sentence.

The words "is a" occur so frequently in English that it is natural to treat them as a single unit and to symbolize them by a binary predicate

symbol. Thus we write

x is a animal \leftarrow x is a human

treating types as individuals rather than as properties of individuals. The treatment of types as individuals increases expressive power. It allows us to write clauses which refer to types by means of variables, for example

x is a y \leftarrow x is a z , z is a y

which expresses the transitivity of "is a". Transitivity cannot be expressed in clausal form if types are treated as properties.

Existence

The English word "some" expresses existence. In the standard form of logic the existence of individuals can be expressed without giving them a name. But in the clausal form of logic, existence is expressed by naming individuals, using constant symbols and function symbols. The sentence

Some men are animals.

for example, can be expressed by means of the clauses

\odot is a man \leftarrow

\odot is a animal \leftarrow

where the constant symbol \odot is not used elsewhere to name a different individual. Notice, however, that the same clauses can also be regarded as expressing the English sentence

Some animals are men.

The English words "has" and "have" often express existence. The sentence

Zeus has a parent who loves him.

for example, can be reexpressed as

Some parent of Zeus loves him.

In clausal form, a constant symbol is needed to name the loving parent. The name doesn't matter provided it is not used elsewhere for a different individual. If the constant symbol \odot satisfies this condition, then the sentence is symbolized by means of the clauses

\odot is a parent of Zeus \leftarrow

\odot loves Zeus \leftarrow

To express that

everyone has a parent who loves him

the loving parent needs to be named by a function symbol. The simpler clauses

is a parent of $x \leftarrow x$ is a human

loves $x \leftarrow x$ is a human

express the stronger assumption that a single individual, who is a parent of everyone, loves everyone. We need to express the more modest assumption that for every human x there is an individual which is a loving parent of x . Different individuals might have different loving parents. The loving parent of x is a function of x and its name needs to be constructed by a function symbol applied to x . Any function symbol can be used, provided it is different from any used elsewhere. If the function symbol "par" satisfies this condition, then the term $\text{par}(x)$ names the loving parent of x and the sentence can be expressed by the clauses

$\text{par}(x)$ is a parent of $x \leftarrow x$ is a human

$\text{par}(x)$ loves $x \leftarrow x$ is a human.

In a similar manner, the assumptions

Everyone has a mother.

Offices have desks.

Birds have wings.

can be symbolized, using function symbols, by such clauses as

$\text{mum}(x)$ is a mother of $x \leftarrow x$ is a human

$\text{d}(x)$ is a desk $\leftarrow x$ is a office

$\text{d}(x)$ is in $x \leftarrow x$ is a office

$\text{w}(x)$ is a wing $\leftarrow x$ is a bird

$\text{w}(x)$ is part of $x \leftarrow x$ is a bird.

Individuals can be named by function symbols having several arguments. The "English" sentence

For every individual x and every list y
there exists a list whose first element
is x and rest is y .

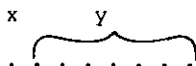
for example, can be expressed by the clauses

$\text{cons}(x,y)$ is a list \leftarrow y is a list

x is the first of $\text{cons}(x,y)$ \leftarrow y is a list

y is the rest of $\text{cons}(x,y)$ \leftarrow y is a list

where the term $\text{cons}(x,y)$ names the list



constructed by putting the element x in front of the list y . Although the infix notation for the clauses is easy to read, the prefix notation is more compact:

$L(\text{cons}(x,y)) \leftarrow L(y)$

$\text{First}(x, \text{cons}(x,y)) \leftarrow L(y)$

$\text{Rest}(y, \text{cons}(x,y)) \leftarrow L(y)$.

The existence of an individual which is referred to in the conclusions of a statement needs to be expressed by a constant symbol or function symbol. However, it needs to be expressed by a variable if the individual is referred to in the conditions of the statement but not in the conclusions. For example

One person is a grandparent of another if
he has a child who is parent of the other.

x is grandparent of $y \leftarrow$ x is human,
 y is human,
 x is parent of $z,$
 z is parent of y

It is often easier to understand a clause if variables which occur in conditions but not in conclusions are read as expressing existence. For example, the clause

Mary likes John \leftarrow Mary likes x

can be read as stating that

if there is anything that Mary likes at all,
then Mary likes John.

The clause

x has $y \leftarrow$ z gives y to x

expresses that x has y if someone gives y to x .

Negation

Negation can be expressed directly in the standard form of logic. In the clausal form it can only be expressed indirectly. The conclusion-less clauses

```
<- Mother(Zeus,x)
```

```
<- Mother(x,y), Father(x,z)
```

for example, state that

```
Zeus is not the mother of anyone and
no one is both a father and a mother.
```

It is a feature of clausal form semantics that a negated condition can be reexpressed as an unnegated conclusion. The sentence

```
Robert is at work if he is not at home.
```

which can be expressed directly with a negative condition

```
At(Robert,work) <- not-At(Robert,home)
```

in standard form can be expressed without negation in clausal form by means of a non-Horn clause

```
At(Robert,work), At(Robert,home) <- .
```

The sentence

```
not-Happy(John) <- not-Likes(Mary,John)
```

in standard form can be reexpressed in clausal form

```
Likes(Mary,John) <- Happy(John).
```

Notice that the different English sentences

```
Every fungus which is not a toadstool is a mushroom.
Every fungus which is not a mushroom is a toadstool.
Everything which is neither a mushroom nor a
toadstool is not a fungus.
```

all have the same clausal form

```
Toadstool(x), Mushroom(x) <- Fungus(x).
```

Denial of conclusions which are implications

In clausal form, to show that assumptions imply a conclusion, it is necessary to deny that the conclusion holds and to demonstrate

inconsistency. A typical conclusion often has the form of an implication:

All boleti are poisonous.

Poisonous(x) \leftarrow Boletus(x)

for example. In general, an implication is a Horn clause with a single conclusion and one or more conditions. A Horn clause with a conclusion, but no condition, is called an assertion. It is often convenient, however, to use the terminology "implication" in the wider sense which includes assertions.

To deny an implication it is necessary to assert the existence of individuals satisfying all of the conditions and to deny that they satisfy the conclusions. In this case, we assert the existence of an individual, say \uparrow , which is a boletus and deny that it is poisonous.

Boletus(\uparrow) \leftarrow

\leftarrow Poisonous(\uparrow)

In Chapter 10, when we investigate the standard form of logic, we shall formulate a systematic procedure for transforming denials of sentences into clausal form. Meanwhile, it suffices to use the rule above for denying conclusions which have the form of implications.

Conditions which are implications

In natural language and in the standard form of logic it is common for a condition to have the form of an implication. For example, the implication

All Bob's students like logic.

which has the structure of a Horn clause

x likes logic \leftarrow x is a student of Bob

is the condition of the sentence

(1) Bob is happy if all his students like logic.

Although the sentence can be expressed directly in the standard form of logic, it needs to be paraphrased before it can be expressed in clausal form. In Chapter 10 we shall present a systematic method for transforming such sentences from standard form into clausal form. Here we can illustrate the method by successively transforming the original sentence (1) in English:

(2) Not all of Bob's students like logic if Bob is unhappy.

(The unnegated condition and conclusion of (1) become the negated conclusion and negated condition of (2).)

- (3) There is a student of Bob, who doesn't like logic, if Bob is unhappy.

(The conclusion of (2), which is the denial of an implication, is reexpressed by asserting the existence of an individual which satisfies the condition of being a student of Bob but not the conclusion of liking logic.)

- (4) There is a student of Bob, say ☹ ,
and ☹ doesn't like logic, if Bob is unhappy.

(The culprit is given a name.)

- (5) ☹ is a student of Bob if Bob is unhappy.
☹ doesn't like logic if Bob is unhappy.

(The two conclusions are expressed by two sentences having the same condition.)

- (6) ☹ is a student of Bob or Bob is happy.
Bob is happy if ☹ likes logic.

(The negated condition is reexpressed as an unnegated conclusion and the negated conclusion as an unnegated condition.)

- (7) ☹ is a student of Bob, Bob is happy <-
Bob is happy <- ☹ likes logic

The transformation from English to clausal form can be compressed. In the simple case where the English sentence has the form

A if B is implied by C.

i.e. A <- [B <- C]

in the standard form of logic, the corresponding clauses have the form

A, C <-

A <- B.

Complications arise when, as in the preceding example, the condition B <- C

contains variables which need to be replaced by constant symbols or terms involving function symbols.

Although sentences having conditions which are implications may appear unnatural in clausal form, they have a natural problem-solving interpretation, discussed in Chapters 7 and 8. In Chapter 10 we shall investigate such sentences in greater detail. Until then we shall concentrate on examples which can be expressed by Horn clauses, whose conditions are simple atomic formulae.

Definitions and "if-and-only-if"

It is normal in mathematics and logic to express definitions by means of "if-and-only-if":

x is grandparent of y if-and-only-if
there is a z which is child of x and parent of y.

The expression

A if-and-only-if B

is interpreted as meaning

A if B and A only-if B.

"A only-if B" is normally interpreted as

B if A.

This interpretation of "only-if", however, is not the only one. In Chapter 11 we shall discuss an alternative interpretation.

The expression "if-and-only-if" can be expressed directly in the standard form of logic. In the clausal form, however, the two halves need to be expressed independently. Moreover, the only-if half is often unnatural. In the case of the only-if half of the grandparent definition

x is parent of rel(x,y) \leftarrow x is grandparent of y
rel(x,y) is parent of y \leftarrow x is grandparent of y

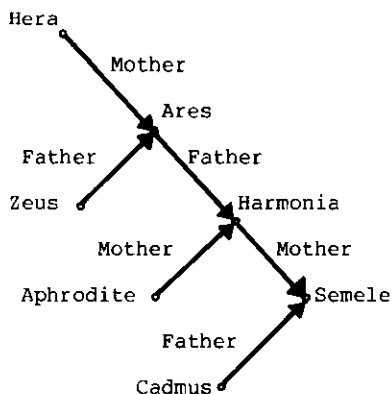
a function symbol is necessary to name the relative of x and y who is a child of x and a parent of y.

If-and-only-if definitions and sentences having conditions which are implications are the two main cases in which clausal form is more awkward than both natural language and the standard form of logic. Until Chapters 10 and 11 we shall avoid complications by using only the if-halves of definitions, which is adequate for most purposes.

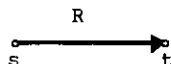
Semantic networks

Many researchers in the field of artificial intelligence use semantic networks, as an alternative to symbolic logic, to represent information in computers. Semantic networks are used both as models of human memory organisation and as representation schemes for the meanings of natural language sentences.

A semantic network is a graph whose nodes represent individuals and whose directed arcs represent binary relationships. Each individual is represented by only one node. The information in the clauses Fl-6 of Chapter 1, for example, can be represented by means of the semantic network.



In general, a semantic network can be regarded as equivalent to the set of variable-free assertions represented by its arcs. An arc labelled R directed from node s to node t



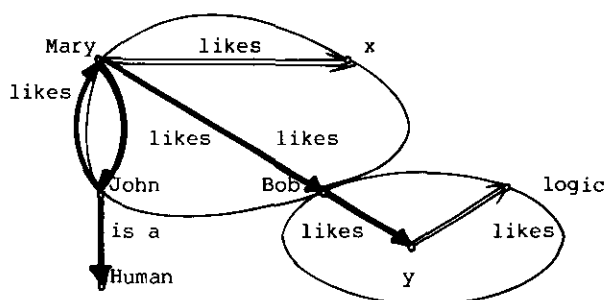
represents the assertion

$$R(s,t) \leftarrow .$$

Simple semantic networks have no provision for representing variables, function symbols, n -ary predicate symbols or clauses having conditions or alternative conclusions. As we shall see later, the restriction to binary relations is not an important limitation, because every n -ary relationship can be reexpressed as the conjunction of $n+1$ binary relationships. Other restrictions, however, are more serious and have motivated several investigators to propose extensions [Shapiro 1971, 1972], [Hendrix 1975], [Schubert 1977], all of which treat semantic networks as an alternative syntax for symbolic logic. The one described below treats extended semantic networks as a pictorial syntax for clausal form [Deliyanni and Kowalski 1979].

Extended semantic networks

As in simple semantic networks, nodes represent individuals and arcs represent binary relationships. However, nodes can be constants, variables or terms constructed using function symbols. Arcs can represent conditions as well as conclusions and are grouped into clauses. Conditions are drawn with two lines and conclusions with one heavy line as before. Clauses containing more than one atom are delimited by enclosing them within subnetworks. The extended semantic network



corresponds to the set of clauses

John likes Mary \leftarrow

John is a human \leftarrow

Mary likes John, Mary likes Bob \leftarrow Mary likes x

Bob likes y \leftarrow y likes logic.

Apart from their pictorial aspect, semantic networks have two other attractions: They provide a useful scheme for storing information, and they enforce the discipline of using binary rather than more general n-ary predicate symbols. The fact that every individual is represented by a single node means that all information about the individual is directly accessible from the node. This feature has been exploited in the design of path-finding problem-solving strategies. In the next two sections, however, we shall compare the use of binary predicate symbols with that of more general n-ary predicate symbols.

The representation of information by binary predicate symbols

Every n-ary relationship can be reexpressed as a conjunction of n+1 binary relationships. For example, the assertion

John gave book to Mary \leftarrow

can be reexpressed in English:

There is an event *e*
 which is an act of giving
 by an actor John
 of an object book
 to a recipient Mary.

In clausal form, ignoring the assertion which describes that *e* is of type "event", the single 3-place relationship can be reformulated as 4 binary relationships.

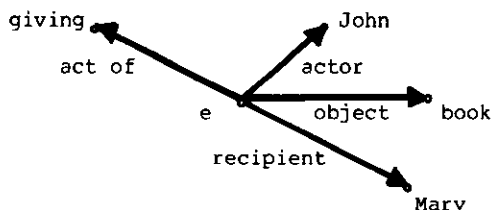
e is an act of giving <-

e has actor John <-

e has object book <-

e has recipient Mary <-

The semantic network representation



of the clauses is similar to the case structure analysis of natural language employed in linguistics [Fillmore 1968] and artificial intelligence [Quillian 1968], [Schunk 1973, 1975], [Simmons 1973].

In general, to replace an *n*-ary relationship by binary relationships it is necessary to treat the *n*-ary relationship and its relation as individuals (giving them names such as "*e*" and "giving" in the preceding example). It is necessary to introduce a binary relationship which expresses that the *n*-ary relationship belongs to the *n*-ary relation: in this example, the binary relationship

e is an act of giving <- .

For every argument of the *n*-ary relationship, a binary relationship is needed to express that the argument belongs to the *n*-ary relationship.

We shall refer to the representation of information by general *n*-ary relationships as the *n*-ary representation and the corresponding representation by means of binary relations as the binary representation.

Binary relationships can replace *n*-ary relationships in both conditions and conclusions of clauses. For example, the English sentence

A person possesses an object
 after it is given to him.

can be expressed in the form

For every event u in which x gives y to z ,
 there exists a situation, say $\text{result}(u)$,
 immediately after u , which is a
 state of possession by the subject z of the
 object y .

The systematic formulation of the sentence in clausal form using binary predicate symbols ignoring types, produces four Horn clauses all having the same conditions.

```

result(u) is immediately after u <- u is an act of giving,
                                     u has actor x,
                                     u has object y,
                                     u has recipient z

result(u) is a state of possession <- u is an act of giving,
                                     u has actor x,
                                     u has object y,
                                     u has recipient z

result(u) has subject z <- u is an act of giving,
                             u has actor x,
                             u has object y,
                             u has recipient z

result(u) has object y <- u is an act of giving,
                             u has actor x,
                             u has object y,
                             u has recipient z
  
```

In this example, the binary representation is less compact than an n -ary representation which includes explicit arguments for the act u and the state $\text{result}(u)$.

```

result(u) is immediately after u <-
                                     u is an act of giving by x of y to z

result(u) is a state of possession by z of y <-
                                     u is an act of giving by x of y to z
  
```

However, if we assume that every act of giving has an actor, object and recipient then the original binary representation can be reformulated more compactly.

```

result(u) is immediately after u <- u is an act of giving
result(u) is a state of possession <- u is an act of giving
result(u) has subject z <- u has recipient z
result(u) has object y <- u has object y
  
```

Advantages of the binary representation

The binary representation is generally more expressive than the n-ary representation. It makes it easier to add new information and to ignore information that is unknown.

In the binary representation, relations and relationships are treated as individuals. Consequently it is possible to talk about them in such sentences as

Mary wants John to give her the book.

Mary wants e <- .

The corresponding expression in the n-ary representation

Mary wants (John gave book to Mary) <-

is not a legal sentence of clausal form.

The ability to talk about relationships in the binary representation also makes it easier to add new information about a relationship. For example, having expressed that

John gave the book to Mary

to add the new information that he did so in Hyde park requires only the addition of a new assertion

Hyde park is the location of e <-

in the binary representation. But, in the n-ary representation, it requires replacing the original assertion which used a 3-place predicate symbol

John gave book to Mary <-

by a new one with a 4-place predicate symbol.

John gave book to Mary in Hyde park <-

Notice, however, that it is really the treatment of relationships as individuals which is responsible for the advantages of the binary representation in the preceding two examples. Both of the sentences

Mary wants e <-

Hyde park is the location of e <-

can be expressed in an n-ary representation with an explicit argument which names the relationship e.

e is an act of giving by John of book to Mary <-

The binary representation is also more convenient than the n-ary



representations when components of a relationship are unknown. For example, to express that

the book was given to John

it suffices in the binary representation simply to state what is known and to ignore what is unknown.

e' is an act of giving <-
 e' has object book <-
 e' has recipient John <-

In the two n-ary representations, on the other hand, it is necessary to give the unknown actor a name.

or  gave book to John <-
 e' is an act of giving by  of book to John <-

The argument in favour of binary relations is not conclusive. There are many relationships, such as

x times y is z,
 x received grade y for course z,
 x is the y-th element of sequence z, and
 v is a proof that the assumptions x
 imply the conclusion y
 obtained by the proof procedure u,

for which an n-ary representation is more convenient than the binary representation. The use of general n-ary relations moreover is more common than the use of binary relations in the field of databases.

Databases

A database is a collection of information to be used for a variety of purposes. A typical database might contain a firm's personnel records, details of bank transactions or the police files of convicted criminals. Increasingly, such databases are represented in a form which can be processed by computers. These are used to update the databases, to check the consistency of data, and to answer requests for information.

A single database might be used to obtain information by many users with little computer training. In this case the data need to be represented in a simple form which is independent of its representation inside the computer. Consequently, the database query language must be both simple to learn and easy to use. It is now widely accepted that these requirements can best be satisfied if data are viewed as relations [Codd 1970].

The relational view of data is equivalent to the representation of data by tables: The argument positions of a relation can be regarded as the columns of a table and the relationships which make up the relation are its rows. Thus the 5-column, 3-row table

Birthday club	Name	Office	Dues	Birthdate	Date joined
	Mary	president	10p	4.Mar.77	4.Mar.77
	John	secretary	10p	2.Mar.78	2.Mar.78
	Bob	treasurer	10p	1.Jan.80	1.Jan.80

represents the 5-argument relation which is described by the 3 assertions:

```
Club(Mary, president, 10p, 4.Mar.77, 4.Mar.77) <-
Club(John, secretary, 10p, 2.Mar.78, 2.Mar.78) <-
Club(Bob, treasurer, 10p, 1.Jan.80, 1.Jan.80) <-
```

The same information can be described by using binary predicate symbols. In this example the binary representation can be simplified because each row of the table can be uniquely identified by the value in its first column. Accordingly, the value in that column is said to be a key of the table. In the binary representation of the table, the key can function as the name of the relationship which it identifies.

```
B1      Member(Mary, birthday club) <-
B2      Member(John, birthday club) <-
B3      Member(Bob, birthday club) <-
B4      Office(Mary, president) <-
B5      Office(John, secretary) <-
B6      Office(Bob, treasurer) <-
B7      Dues(Mary, 10p) <-
B8      Dues(John, 10p) <-
B9      Dues(Bob, 10p) <-
B10     Birthdate(Mary, 4.Mar.77) <-
B11     Birthdate(John, 2.Mar.78) <-
B12     Birthdate(Bob, 1.Jan.80) <-
B13     Datejoined(Mary, 4.Mar.77) <-
B14     Datejoined(John, 2.Mar.78) <-
B15     Datejoined(Bob, 1.Jan.80) <-
```

Notice that the binary representation of the table, though more longwinded, is easier to read than the n-ary representation. The names of the columns, which are necessary for understanding the table, are not represented in the n-ary representation, but are represented by binary predicate symbols in the binary representation.

More importantly from a computational point of view, the binary representation can often express general laws which could not be expressed at all in the n-ary representation. In particular, the general laws

```
Dues(x, 10p)      <- Member(x, birthday club)
Datejoined(x,y)   <- Member(x, birthday club),
                    Birthdate(x,y)
```

can replace the specific assertions B7-9 and B13-15 in the binary representation, but cannot be formulated in the n-ary representation at all.

Data query languages

The relational view of data has been used more for data queries than for data description.

Most relational query languages use the symbolism of symbolic logic or relational algebra. Relational calculus query languages [Codd 1972] can be regarded as using a binary representation of relations. Given, for example, the data contained in the Birthday club and the Address tables

Birthday club	Name	Office	Dues	Birthday	Date joined

Address	Name	Street number	Street	Town

the query What Birthday club members live
 on Euclid Avenue?

can be formulated in the binary representation

```

<- Answer(x)
Answer(x) <- Member(x, birthday club),
             Street(x, Euclid Ave)

```

in a manner similar to that of the relational calculus. It can also be formulated in the n-ary representation

```

<- Answer(x)
Answer(x) <- Club(x,y,z,u,v),
             Address(x, y', Euclid Ave, z')

```

similar to that of the tabular query-by-example language [Zloof 1975].

The relationship between queries expressed in the clausal form of logic and ones expressed in query-by-example has been investigated by van Emden [1979]. A classification of relational query languages, all based on the standard form of logic, has been made by Pirotte [1978].

Data description

The relational model of data is not concerned with the formalism used to represent data within the computer. It is compatible with any formalism which can be viewed abstractly in terms of relations. Nevertheless, the use of symbolic logic is especially attractive. It has the advantage that the same formalism can be used both for expressing queries and for defining data. Moreover, when the data can be defined by

means of general laws, the data definitions are indistinguishable from programs. The sentence

```
Dues(x, l0p) <- Member(x, birthday club)
```

for example, can be regarded both as a general law and as a program which computes the dues paid by members of the birthday club.

Symbolic logic was used before the relational model of databases to describe both data and queries in question-answering systems. Among the first systems were those described by Darlington [1969] and Green [1969a, 1969b]. The use of the "Answer" predicate symbol, in particular, was introduced by Green. More recent systems have been developed in Marseille [Colmerauer et al 1972], [Dahl and Sambuc 1976] and Maryland [Minker et al 1973], [McSkimin and Minker 1977], and by Nicolas and Syre [1974] and Kellogg, Klahr and Travis [1978].

Integrity constraints

Since data often contain errors, integrity constraints are used to describe properties which the data need to satisfy in order to be correct. The clause

```
y is before z <- Today(z),
                    Member(x, birthday club),
                    Birthdate(x,y)
```

for example, expresses that all members of the birthday club were born before today. If today were 1.Apr.79

```
Today(1.Apr.79) <-
```

then given an appropriate definition of the is before relation, the data

```
Member(Bob, birthday club) <-
Birthdate(Bob, 1.Jan.80) <-
```

would be inconsistent with the integrity constraint and should be rejected by an intelligent database management system.

Using symbolic logic as a formalism for describing information blurs the conventional distinction between databases and programs. Integrity constraints for databases are indistinguishable from program properties. The clause

```
x ≤ y <- Fact(x,y)
```

for example, describes a property which needs to be satisfied by a correct definition of the factorial relation. Like an integrity constraint, its purpose is not to contribute to the definition of the ≤ and Fact relations but rather to constrain the definitions from having unacceptable properties.

Integrity constraints can be used for other purposes. They can be used to reject inconsistent queries

What number is less than 1,300
and is the factorial of 5,200 ?

and to transform difficult goals into easier ones. The use of integrity constraints to aid problem-solving is investigated in Chapter 9.

A departmental database

The PROLOG [Roussel 1975] Horn clause problem-solving system developed in Marseille has been used for a variety of tasks which combine features of both databases and programs. It has been used in Marseille for natural language question answering [Colmerauer et al 1972], [Dahl and Sambuc 1976] and symbolic integration [Bergman and Kanoui 1973], in Edinburgh for plan-formation [Warren 1974, 1976], geometry theorem-proving [Welham 1976], [Coelho and Pereira 1975], the solution of mechanics problems expressed in English [Bundy et al 1979] and compiler-writing [Warren, Pereira and Pereira 1977] and in Budapest for computer-aided design [Markusz 1977] and drug analysis [Futo, Darvas and Szeredi 1978]. In London we have implemented part of a database which describes the activities of our department. The following clauses are typical of those used to describe the data.

```

x is occupied with y <- x teaches y
x is occupied with y <- x attends y
x is occupied with y <- x is member of committee y
9:30 is the hour of 304 <-
Fri is the day of 304 <-
3 is the level of 304 <-
145 is the room of x <- 3 is the level of x
RAK teaches 304 <-
145 has capacity 80 <-
65 people attend 304 <-
x attends y <- x is a student in year z,
                        z is the level of y
Problem-solving is the name of 304 <-

```

Here it is assumed that course 304 meets only once a week. If it meets more often, then composite terms, `part(304,1)`, `part(304,2)`, for example, might be used to name different parts of the course.

Various integrity constraints, such as

```

<- x is the room of y, x has capacity u,
    v people attend y, u < v

```

can be expressed and tested for consistency with the data. Queries can be answered by denying that they have an answer, proving inconsistency and extracting from the proof the information needed to construct the answer. Thus, to determine the activity with which RAK is occupied at 9:30 on Fridays it suffices to deny that there is such an activity:

```

<- Answer(x)
Answer(x) <- RAK is occupied with x,
              9:30 is the hour of x,
              Fri is the day of x

```

The substitution $x = 304$

which can be extracted from the proof answers the query. The answer extraction can be done automatically by the problem-solving system.

Equality

Mathematical notation normally uses function symbols and the binary predicate symbol = (equality) where we have used other predicate symbols. It is usual to write

$x * y = z$	instead of	$\text{Times}(x, y, z)$
$x! = y$	instead of	$\text{Fact}(x, y)$
$x = \text{father}(y)$	instead of	$\text{Father}(x, y)$

Similarly, the relational calculus query language uses function symbols and equality, writing

$\text{office}(x)$	$= y$	instead of	$\text{Office}(x, y)$
$\text{dues}(x)$	$= y$	instead of	$\text{Dues}(x, y)$
$\text{birthdate}(x)$	$= y$	instead of	$\text{Birthdate}(x, y)$
$\text{datejoined}(x)$	$= y$	instead of	$\text{Datejoined}(x, y)$

Functional notation is often more compact than relational notation. It is simpler, for example, to express

The date on which a member of the
birthday club joins the club is the
same as his birth date.

in the functional notation

$\text{birthdate}(x) = \text{datejoined}(x) \leftarrow \text{Member}(x, \text{birthday club})$

than in the relational notation

$\text{Birthdate}(x, y) \leftarrow \text{Member}(x, \text{birthday club}), \text{Datejoined}(x, y)$
 $\text{Datejoined}(x, y) \leftarrow \text{Member}(x, \text{birthday club}), \text{Birthdate}(x, y)$.

Equality is necessary whenever an individual has more than one name. For example:

$\text{Jove} = \text{Jupiter} \leftarrow .$

It is also necessary, even in the relational notation, to express that one argument of a relation is a function of the others. For example:

$x = y \leftarrow \text{Father}(x, z), \text{Father}(y, z)$

To show that a set of clauses S containing the equality symbol is inconsistent, the set of clauses needs to contain the following axioms characterising the equality relation, for every function symbol f and every predicate symbol P occurring in S , (including the equality symbol).

```

E1      x = x <-
E2      P(x1, ..., xm) <- P(y1, ..., ym), x1=y1, ..., xm=ym
E3      f(x1, ..., xm) = f(y1, ..., ym) <- x1=y1, ..., xm=ym

```

For example, to demonstrate that the assumptions

```

J1      Jekyll = Hyde <-
J2      father(John) = Hyde <-
J3      Member(father(John), birthday club) <-

```

imply the conclusion

```

Member(Jekyll, birthday club) <-

```

it is necessary to deny the conclusion

```

J4      <- Member(Jekyll, birthday club)

```

and add the appropriate axioms for the equality relation:

```

J5      x = x <-
J6      Member(x1, x2) <- Member(y1, y2), x1 = y1, x2 = y2
J7      x1 = x2 <- y1 = y2, x1 = y1, x2 = y2
J8      father(x) = father(y) <- x = y

```

The resulting set of clauses J1-8 is inconsistent because J1-3 are "obviously" inconsistent with the instances

```

Hyde = Hyde <-
birthday club = birthday club <-
Member(Jekyll, birthday club) <- Member(father(John), birthday club),
                                   Jekyll = father(John),
                                   birthday club = birthday club
Jekyll = father(John) <- Hyde = Hyde, Jekyll = Hyde, father(John) = Hyde

```

of J5-7. Clause J8 in this example does not contribute to the inconsistency.

Problem-solving is considerably simplified if individuals have only one name (distinct variable-free terms naming distinct individuals). Then the single axiom

```

E1      x = x <-

```

expresses the only situation in which two individuals are the same (if they have the same names). The infinitely-many axioms

```

D      Diff(s, t) <-

```

for every pair of distinct variable-free terms s and t , express the only situations in which individuals are different (if they have different names). Given a finite set of clauses S the infinitely-many axioms D can be replaced by finitely many clauses

- a) Everyone likes someone.
- b) Everyone likes everyone.
- c) Someone likes everyone.
- d) No one likes anyone.
- e) No one likes someone.
- f) Someone likes no one.
- g) John and Mary like themselves.
- h) A teacher is happy if he belongs to no committees.
(Paraphrase the sentence first: It is not the case that a teacher is happy and belongs to some committee.)
- i) Anyone who knows anything about logic likes logic.

2) In each of the following arguments the assumptions imply the conclusion. Express the assumptions and the denial of the conclusion in clausal form, so that the resulting set of clauses is inconsistent. Demonstrate inconsistency by showing that the set of clauses is true in no interpretation.

- a) Assumption There is a single individual who is a loving parent of everyone.

Conclusion Everyone has a parent who loves him.

- b) Assumptions All easterners like all westerners.
All westerners like all easterners who like some westerner.

Conclusion All westerners like all easterners without exception.

- c) Assumptions Canaries are birds.
All birds have wings.

Conclusion Canaries have wings.

- d) Assumptions Anything which accomplishes something good is good itself.
Anything which accomplishes something bad is bad itself.
War accomplishes both peace and suffering.
Peace is good and suffering is bad.

Conclusion Some things are both good and bad.

- e) Assumptions x is a member of $\text{cons}(x,y)$.
 x is a member of $\text{cons}(u,y)$ if x is a member of y .

Conclusion A is a member of $\text{cons}(C, \text{cons}(A, \text{cons}(C, \text{nil})))$.

- f) Assumption Bob is happy if all his students like logic.

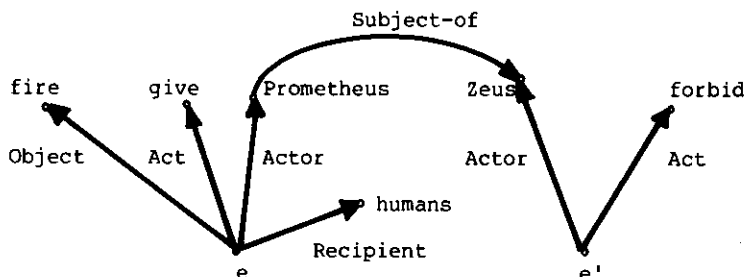
Conclusion Bob is happy if he has no students.

3) The word "like" in exercise (6) of Chapter 1 disguises two different meanings. Redo exercise (6) distinguishing between the notions

x likes to eat y and
x likes to be with y.

You can do so either by using two completely distinct predicate symbols, $Like_1$ and $Like_2$, or by using a single three argument predicate symbol, one of which is the name of an event (eating) or of a state (being with).

4) Express in clausal form the information represented in the following semantic network and English sentences:



The object of e' is any act of giving fire to humans. If a ruler forbids an act which is performed by one of his subjects then there is another event in which the ruler punishes the subject.

5) This exercise is based on Schank's [1973, 1975] conceptual analysis of actions. Let the intended interpretation of

Act(x,y)	be	x is an act of type y,
Possess(x,y,u)		x possesses y in state u,
Actor(x,y)		the actor of act x is y,
Object(x,y)		the object of act x is y,
Donor(x,y)		the donor of act x is y,
Recipient(x,y)		the recipient of act x is y.

Let the terms

ATRANS	name	the type of all acts of abstract transactions,
GIVE		the type of all acts of giving,
TAKE		the type of all acts of taking,
result(u)		the state immediately after the act u,
prior(u)		the state immediately prior to the act u.

Express the following sentences in clausal form:

- a) In the state immediately after any act of type ATRANS, the recipient of the act possesses the object of the act.
- b) In the state immediately prior to any act of the type ATRANS, the donor possesses the object of the act.
- c) An act of type ATRANS is an act of giving if the actor is the donor.
- d) An act of type ATRANS is an act of taking if the actor is the recipient.

6) Redo exercise (5) using equality and function symbols. Let

act(x)	name	the type of act x,
actor(x)		the actor of x,
object(x)		the object of x,
donor(x)		the donor of x,
recipient(x)		the recipient of x.

7) Let Parents(x,y,z) hold when x is the father and y the mother of z. Formulate a set of clauses whose only variable-free assertions concern the Parents relation but which imply the variable-free assertions Fl-8 of Chapter 1.

8) Assume that data is given in the Supplier, Part and Supply tables:

Supplier	Supplier-Number	Name	Status	City

Part	Part-Number	Name	Colour	Weight

Supply	Supplier-Number	Part-Number	Quantity

Formulate the following queries in clausal form. Use both the binary and the n-ary representations, taking advantage of the fact that Supplier-Number is a key of the Supplier table and Part-Number is a key of the Part table. Assume that the relationship

$x < y$ (x is less than y)

is already given.

- a) What are the numbers of suppliers of nuts?
- b) What are the names of suppliers of bolts?
- c) What are the locations of suppliers of nuts and bolts?
- d) What are the names of parts supplied by the supplier named John?
- e) What are the names of suppliers located in London who supply nuts weighing more than one ounce?
- f) What are the names of suppliers of both nuts and bolts?
- g) What are the names of suppliers of nuts or bolts?